ISSN 0869-3889

МОНИТОР

Версия 1.91

Устройство памяти IBM PC

Объектно–ориентированное программирование

DOS под микроскопом

Создание резидентных программ

Этюды для программистов

Компьютерный бизнес в СССР

N 1

DOS

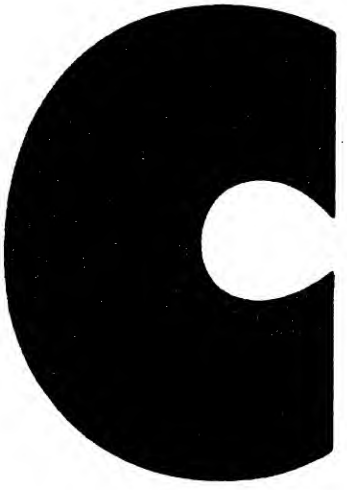
IBM PC

Все о TURBO C и C++!

"Библиотека журнала "Монитор"
представляет первую книгу

"Функции Turbo C, C++".

Более чем на 300 страницах тщательно подобрана информация о библиотеках C, Turbo C, C++.



Вы получите:

- детальное **описание более 400 функций;**
- примеры, поясняющие работу **каждой функции;**
- список совпадающих функций Turbo C, C++, Microsoft C;
- демонстрационные программы, поясняющие работу групп функций;
- указатели для быстрого поиска по имени функции или ее назначению;
- введение в объектно-ориентированные средства Turbo C++;
- современный дизайн и твердый переплет;

Эта книга может служить справочником и учебником. Мы позаботились о том, чтобы Вам было удобно и приятно ее читать.

Если вы программируете на Turbo C и C++, интересуетесь вопросами совместимости с Microsoft C, то эта книга для Вас.

Вышлите 3 экз. кни
„Функции Turbo C

ПОЧТОВАЯ КАРТОЧКА

Куда 117419, г. Москв
а/я 765

Кому „Монитор"

117419

Пишите индекс предприятия связи места назначен

Спешите! Тираж ограничен.

Книга рассылается наложенным платежом. Выход в свет - декабрь 1991 г. Цена 12 руб. 50 коп. Оптовым покупателям предоставляется скидка. Заказы направляйте по адресу: 117419, Москва, а/я 765, "Монитор", тел. (095)-236-96-13

ПП "Офсет" МГПО. Зак. 1208. Тир. 10 000

МОНИТОР

В середине 80-х годов, с удивлением разбуженного среди ночи, страна внимала речам нового лидера. Прозвучали и вошли в лексикон, став самыми модными, новые слова: перестройка, ускорение, компьютеризация ... По-разному сложилась их судьба. Одни постепенно вышли из употребления, другие же изменились, приобрели новый смысл.

Компьютеры стали ассоциироваться скорее с бизнесом, нежели с наукой или производством. "Доллары-компьютеры-рубли" - наша отечественная интерпретация известной схемы "деньги-товар-деньги".

Можно с уверенностью сказать, что при зарождении новых экономических отношений компьютеры выполнили роль катализатора. Однако эффект от появления компьютеров в цехах и лабораториях оказался меньше ожидаемого.

Стало очевидным, что новая техника, какой бы "умной" она не была, сама по себе не решает проблем, стоящих перед экономикой и наукой. Чтобы эффективно ее использовать, нужны люди - специалисты высокого класса. А для их подготовки необходимы развитые общественные структуры: система образования и повышения квалификации, разветвленная информационная сеть, периодические издания - газеты и журналы.

Надеемся, что в создание таких структур внесет свой вклад и наш журнал. Название журнала выбрано не случайно. Изначальный смысл латинского слова "monitor" - "тот, кто советует, напоминает, предостерегает". Наша задача - помочь всем, кто использует в своей работе персональный компьютер, стать надежным советником и проводником на пути овладения профессиональным мастерством.

Наши принципы.

1) Практическая направленность.

Большинство статей будет сопровождаться текстами программ, которые Вы сможете использовать в своей работе.

2) Ориентация на отечественных программистов и их проблемы.

Мы будем писать о том, что интересно нашим программистам сегодня и возможно завтра. Постоянная связь с читателем даст журналу шанс всегда быть актуальным и интересным.

3) Не только журнал, но и информационная сеть обратной связи "читатель-журнал-читатель".

В наши планы входит:

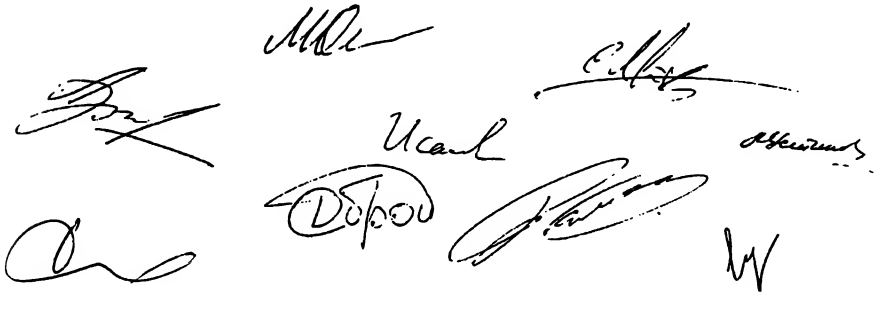
- выпуск электронного приложения на дискетах "Монитор-Диск" с полными текстами программ, освещаемых в журнале;
- издание книжного приложения "Библиотека журнала "Монитор" (первая книга - "Функции Turbo C, C++" выйдет в конце 1991 года);
- проведение читательских семинаров и конференций.

Наряду с проблемными статьями в "Мониторе" будет много интересной информации о развитии мировой компьютеризации, о последних достижениях в области программирования и аппаратуры, о перипетиях компьютерного бизнеса. Найдется место для юмора и веселых историй. Наконец, как любой современный журнал, "Монитор" предоставит свои страницы для рекламы.

Если "Монитор" будет лежать на Вашем рабочем столе истрепанный, ощетинившийся закладками и залитый кофе - значит мы на верном пути.

Но путь этот трудный, и мы рассчитываем на Вашу поддержку, уважаемый читатель. Успех журнала в наших и Ваших руках. Читайте, критикуйте, пишите статьи и письма в редакцию и помогайте нам создавать наш общий журнал.

Редколлегия



Всесоюзный журнал для программистов-практиков

Редколлегия:

Д.Э.Бройтман

И.Г.Гун
(главный редактор)

И.Н.Исаев

С.В.Макельский
(главный художник)

А.В.Орехов
(издательский директор)

М.В.Остринский

С.М.Рамазанов

Л.И.Устинова

А.А.Якубенко
(ответственный секретарь)

Подписано в печать с оригинал-макета 01.09.91. Формат 60x90/8. Печать офсетная. Печ.л.6.0. Тираж 10 000 экз. Изд. N1971. Цена договорная.

Издатель:
Научно-техническое общество

Софт·Москва

Для писем: 117419, г.Москва, а/я 765, тел. 236-96-13

© "Монитор",1991

Хозрасчетное предприятие

270023, Одесса,
ул. Пушкинская, 77.
Телефоны:
66-84-62, 25-04-77
Телекс: 232144 REZON SU
р/с 609070 в Жовтневом отд.
ЖСБ г. Одессы,
МФО 328038

Программное обеспечение хозрасчетного предприятия "Интех" - это ресурсосбережение

"Интех-Раскрой" -система автоматизированного проектирования карт раскроя (КР) и управляющих программ (УП) для машин термической резки (МТР) с числовым программным управлением (ЧПУ).

Позволяет:

- автоматизировать разводку инструментов для осуществления многорезаковой обработки;
- проектировать КР в автоматическом и интерактивном режимах;
- назначить положение точек врезки, порядок и направление обхода контуров деталей, с учетом технологических требований в ручном и автоматическом режимах;
- осуществлять вывод КР на графические устройства;
- генерировать УП для любых систем ЧПУ.

"Интех-Верификатор" - система автоматизированной отладки, проверки и корректировки УП для МТР с ЧПУ.

Позволяет:

- контролировать семантику и синтаксис УП;
- визуализировать процесс резки с моделированием тепловых полей;
- автоматически подсчитать время резания и координаты инструмента в любой точке обрабатываемого контура;
- изменять порядок обработки контуров деталей, положение точек врезки и их длину.

"Интех-Пресс" - система автоматизированного проектирования технологии обработки деталей и получения УП для прессов с ЧПУ.

Позволяет:

- рационально размещать детали на листе;
- осуществлять оптимальный выбор инструмента для обработки контуров любой сложности с учетом технологических требований;
- генерировать УП для любых систем ЧПУ.

Программно-аппаратные средства, которые могут быть установлены в любой свободный системный разъем ПЭВМ:

- адаптеры сопряжения IBM PC/XT/AT с перфорационными станциями СМ-6204, ПЛ-150 и фотосчитывателем FS-1501.

"Интех-Банк" - система автоматизации банковских операций. Имея опыт внедрения "ИНТЕХ-БАНК" в крупных банках СССР, мы готовы в кратчайший срок, с учетом специфики банка, скомпоновать из требуемых модулей и установить необходимую конфигурацию системы, помочь в ее адаптации и обучить персонал банка.

Предлагаем к приобретению следующие модули:

1. Подготовка, накопление, статистическая обработка кассовых планов.
2. Статистическая обработка отчетов об обороте кассы.
3. Учет документов по расходу кассы.
4. Учет документов по приходу кассы.
5. Расчеты по вкладным книжкам военнослужащих.
6. Расчеты по факторингу.
7. Составление баланса по внебалансовым счетам.
8. Учет документов, ожидающих акцепта.
9. Учет документов, не оплаченных в срок.
10. Автоматизация процедур открытия и закрытия счетов.
11. Учет основных средств.
12. Шифровка телеграмм.
13. Курьер (прием данных с рабочих мест, подготовка макетов с ИВЦ, автоматический обмен данными с ИВЦ по настраиваемому протоколу).
14. Ведение нормативной базы банка.

Наше оборудование позволит Вам связываться с пользователями телеграфных и телексных линий непосредственно с персонального компьютера. Сегодня наша продукция применяется в банковских учреждениях, в вычислительных центрах, ведомственных центрах коммуникации сообщений, учреждениях и оффисах - практически во всех областях, где нужен быстрый, удобный и эффективный доступ к телеграфным сетям

Сегодня мы предлагаем:

- EXT/04 - телексный контроллер для двухпроводных телеграфных линий.
- EXT/05 - телексный контроллер для четырехпроводных телеграфных линий.
- EXT/03 - устройство дежурного ("ночного") приема телеграмм.
- EXT/06 - конвертер интерфейса RS232C в RS423A(RS422A).
- EXT/07 - интеллектуальный 8-ми канальный концентратор для телеграфных, телефонных и выделенных линий.

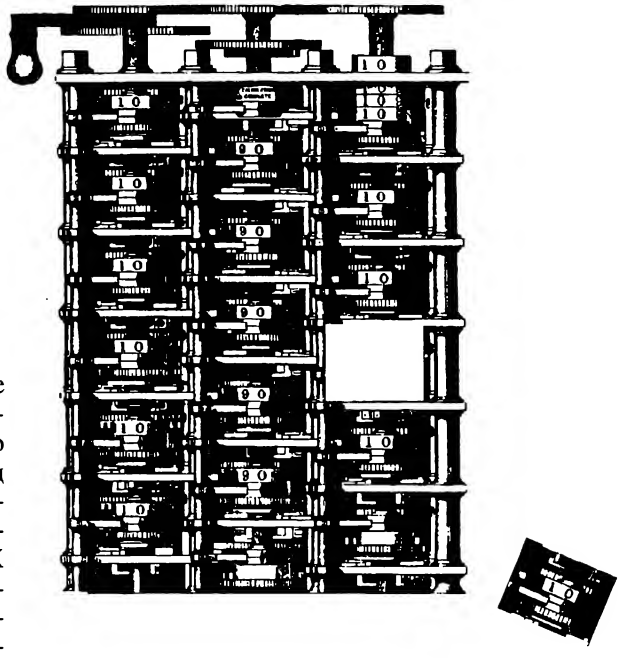
Содержание

Компьютер в разрезе	А. Тихонов Что-то с памятью моей стало ...	4
Технологии программирования	М. Черкашин Объектно-ориентированное программирование: что это такое и для чего оно нужно	12
Экспертные системы	И.Исаев Сколько экспертных систем уместится на кончике иглы?	23
DOS под микроскопом	М.Остринский Создание резидентных программ и работа с прерываниями	29
	В.Аношин Управление загрузкой ОС на IBM PC	36
Панорама	История мировой компьютеризации	42
Этюды для программистов	Д.Бройтман Беспорядок в строю или анатомия одного цикла	44
Компьютерный бизнес	С.Рамазанов Компьютерный бизнес в СССР	47
На досуге	Д.Кузнецов Неуловимый Чуча	49

ЧТО–ТО С ПАМЯТЬЮ МОЕЙ СТАЛО...

А.Тихонов

(Часть I)



В один не очень прекрасный день я обнаружил, что на моей стандартной AT с одним мегабайтом опять не хватает памяти

Когда в 1986 году я впервые начал работать на ПЭВМ, совместимой с IBM PC, мне всего хватало. Действительно, перед этим я в течение двух лет разрабатывал графическое программное обеспечение на ДВК с 64 Кбайтами оперативной памяти, и голова у меня постоянно болела от нехватки двух сотен байт. Теперь мои ресурсы увеличились в десять раз, и я мог позволить себе роскошь хранения часто используемых графических образов непосредственно в оперативной памяти. Но время шло, я сменил две ПЭВМ и в один не очень прекрасный день обнаружил, что на моей стандартной AT с одним мегабайтом опять не хватает памяти.

Давайте попробуем ответить на исконно русские вопросы об американской технике: что же произошло, кто виноват и что же делать, а иными словами, почему увеличились программы, почему нельзя механически поставить дополнительные блоки памяти, и как справиться с дефицитом оперативной памяти.

Начнем с истории. В 1978 году фирма Intel выпустила в продажу первый шестнадцатиразрядный микропроцессор (МП) 8086, а в 1979 МП 8088. МП 8086 и 8088 полностью программно совместимы и отличаются только тем, что 8088 имеет не шестнадцати-, а восьмиразрядную шину данных.

В 1981 году на основе МП 8086 фирма IBM выпустила свою первую ПЭВМ IBM PC, начавшую победное шествие на рынке вычислительных средств. Специально для нее фирмой Microsoft была разработана операционная система, которую Microsoft именует MS DOS, IBM - PC DOS, а мы в дальнейшем будем именовать просто ДОС. Здесь мы не останавливаемся на отличиях MS и PC DOS, достаточно сказать, что PC DOS работает только на ПЭВМ, полностью совместимых с IBM PC. К ним, кстати, относятся отечественные ПЭВМ ЕС-1840...ЕС-1845. Принципиальным отличием архитектуры IBM PC является открытость, что обусловило с тех пор выпуск миллионов ПЭВМ как самой IBM, так и несметным числом ее известных и малоизвестных конкурентов во всем мире. Более 90

процентов этих ПЭВМ работают под управлением ДОС.

В свою очередь с ДОС оказались связанными миллионы прикладных и системных программ, а сама ДОС - с архитектурой микропроцессоров 8086/8088.

Зачем нужно так много памяти

За удобство и функциональность всегда приходится платить сложностью. Действительно, если широко распространенной и знаменитой третьей версии Turbo Pascal фирмы Borland International требовалось всего 64 Кбайта оперативной памяти, и работала она под управлением не только MS DOS, но и более старой операционной системы CP/M, то шестая версия требует для работы все 640 Кбайт ПЭВМ, да, кроме того, пытается использовать дополнительную память, если она,конечно, есть. Шестая версия гораздо удобнее третьей, содержит на порядок больше возможностей, позволяет легко и просто строить почти профессиональные пользовательские интерфейсы с использованием меню, диалоговых окон, кнопок и мышки.

Прикладные пользовательские программы, выпущенные в 1990-1991 годах и выполняемые под управлением ДОС, также требуют ресурсов памяти, превышающих 640 Кбайт. Так, новая версия известнейшей электронной таблицы Lotus 1-2-3 требует теперь двух мегабайт оперативной памяти. Работать с пользовательскими программами на AutoLisp для системы AutoCad невозможно при наличии только 640 Кбайт. Графическая среда Windows-3, установленная на ПЭВМ с одним мегабайтом памяти, производит достаточно тяжелое впечатление. Для того чтобы Borland Pascal, работающий в среде Windows, транслировал программу длиной в пару десятков строк, нужен еще один мегабайт оперативной памяти. Попробуем теперь разобраться в следующем.

Как устроена оперативная память у МП 8086

МП 8086 представляет собой шестнадцатиразрядный МП

МП 8086 представляет собой шестнадцатиразрядный МП, но адресное пространство его составляет один мегабайт (1 048 576 байт). Для адресации используются два слова, старшее из которых называется селектором сегмента, а младшее - смещением. Селектор сегмента позволяет адресовать память с точностью до 16 байт (параграф), а с помощью смещения адресуются 64 килобайта внутри сегмента. Длина адреса МП 8086 составляет 20 бит, а его формирование осуществляется следующим образом: селектор сегмента сдвигается влево на четыре бита и складывается со смещением, что проиллюстрировано на рис. 1. Фактически для адресации используется 32-битовое слово, в котором задействовано всего 20 бит.

Остается только догадываться, что толкнуло инженеров фирмы Intel на использование сегментной схемы адресации, но это решение заставило их сильнейшим образом перерабатывать архитектуру МП 80286 и 80386 с тем, чтобы, с одной стороны, сохранить совместимость с 8086, а с другой - расширить адресное пространство. Рассмотрим теперь устройство адресного пространства IBM PC.

Разработчики архитектуры IBM PC предоставили под операционную систему и прикладные программы 640 Кбайт

Магические 640 Кбайт

Разработчики архитектуры IBM PC предоставили под операционную систему и прикладные программы 640 Кбайт (шестнадцатиричные адреса 0080H...0FFFH), зарезервировав верхнюю часть адресного пространства, начиная с шестнадцатиричного адреса A000H, для использования постоянным запоминающим устройством (ПЗУ), базовой системой ввода/вывода (BIOS), графическими платами и другими аппаратными средствами. Типичное распределение памяти в IBM PC представлено на рис. 2. Объем доступной пользователю памяти (M) в килобайтах определяется таким образом по формуле

$$M = 640 - D - R - T,$$

где D - область памяти, используемая ядром ДОС, системными таблицами; R - память, занимаемая драйверами (сетевой драйвер, драйвер мыши ...); T - постоянно присутствующие в памяти пользовательские программы (TSR).

Совсем необязательно, чтобы в ПЭВМ физически присутствовал весь мегабайт оперативной памяти. Первые IBM PC выпускались всего с 128 Кбайтами оперативной памяти. То же касается и верхней части адресного пространства.

Область адресов с A000H по BFFFH зарезервирована для видеопамяти, но используется только частично. Так, первые IBM PC, выпускавшиеся с монохромным алфавитно-цифровым дисплейным адаптером (MDA), использовали только 4000 байт (80*25=2000 символов на экране плюс столько же для хранения атрибутов каждого символа) в диапазоне B000H...B100H. Цветной графический адаптер нуждался уже в 16 килобайтах (640 * 200 пикселей) и занимал адреса B800H...BFFFH. Графические адаптеры EGA и VGA используют область памяти B800...C000h в текстовом ре-

жиме, а также в режимах эмуляции CGA. В графическом режиме используется область A000H...BFFFH. Область адресов больших F000H также резервируется разработчиками аппаратуры. Так, на оригинальных IBM PC AT в памяти по адресам F600H...FC000H расположено ПЗУ с интерпретатором языка Basic, а в PS/2 фирмы IBM верхние 128 Кбайт занимает системное ПЗУ.

Таким образом, магическое число 640 Кбайт определяется, с одной стороны, архитектурой МП 8086, а с другой - конструктивными решениями, принятыми разработчиками IBM PC. Первое из этих ограничений снято в следующей модели семейства микропроцессоров 8086/8088.

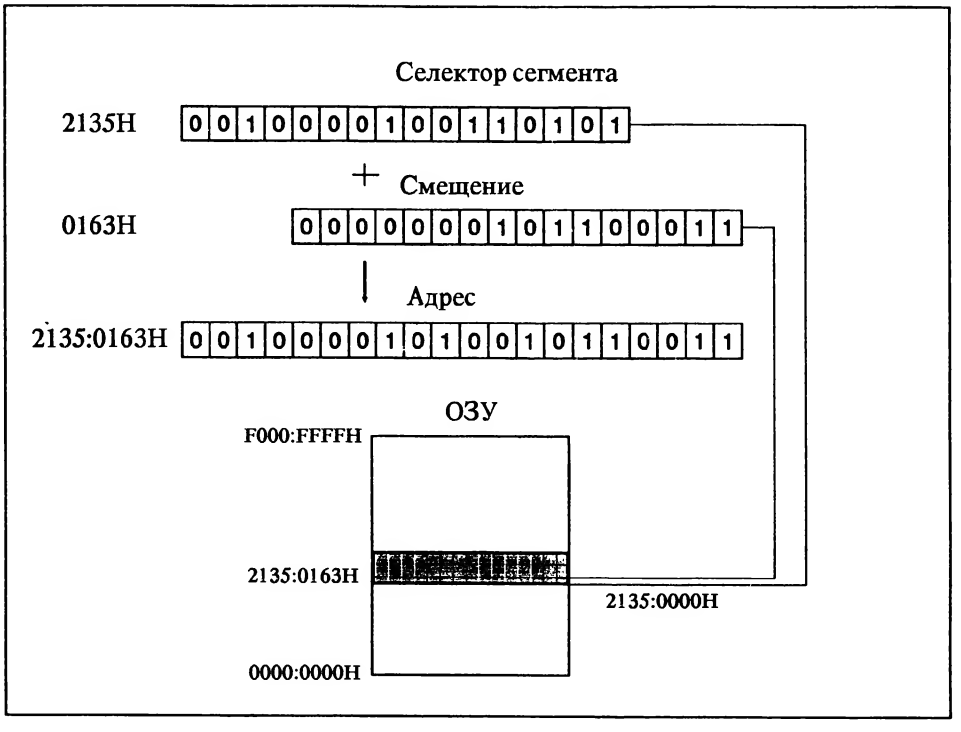
Расширенная память МП 80286

В 1983 году фирма Intel выпустила новый МП 80286, а годом позже IBM объявила о разработке новой модели ПЭВМ: PC/AT. Конструкторы микропроцессора более, чем вдвое, увеличили его производительность по сравнению с 8086 и в 16 раз увеличили максимально возможный размер оперативной памяти, длина адреса стала 24 бита. Увы, инженеры фирмы Intel еще не представляли, как сильно они связаны с IBM PC и ДОС, иначе скорее всего они приняли бы совсем другие архитектурные решения.

МП 80286 при включении питания работает как 8086/8088. Все программы, разработанные для 8086, выполняются на 80286, но в этом режиме МП позволяет адресовать только 1 Мбайт.

Использование полного адресного пространства возможно только в так называемом защищенном (protected) режиме [1]. Название режима связано с тем, что МП аппаратно обеспечивает защиту памяти для нескольких задач, выполняющихся на нем одновременно в этом режиме. В защищенном режиме 16-битовый селектор определяет базовый адрес сегмента, однако схема адресации в значительной степени изменилась (рис. 3). Старшие 13 бит селектора сегмента теперь являются индексом таблицы, содержащей 24-битовые адреса сегментов. В зависимости от значения бита Ti выборка адреса сегмента осуществляется либо из глобальной таблицы дескрипторов (GDT), либо из локальной таблицы (LDT), связанных с текущей задачей. Память, расположенная выше одного мегабайта, получила название расширенной (extended).

Рис. 1. Механизм формирования адреса в МП 8086



Таким образом, PC AT может адресовать до 16 Мбайт памяти. В ПЭВМ, выпущенных в последние годы, на системной плате расположен 1 Мбайт оперативной памяти, из них, 384 килобайта являются расширенными.

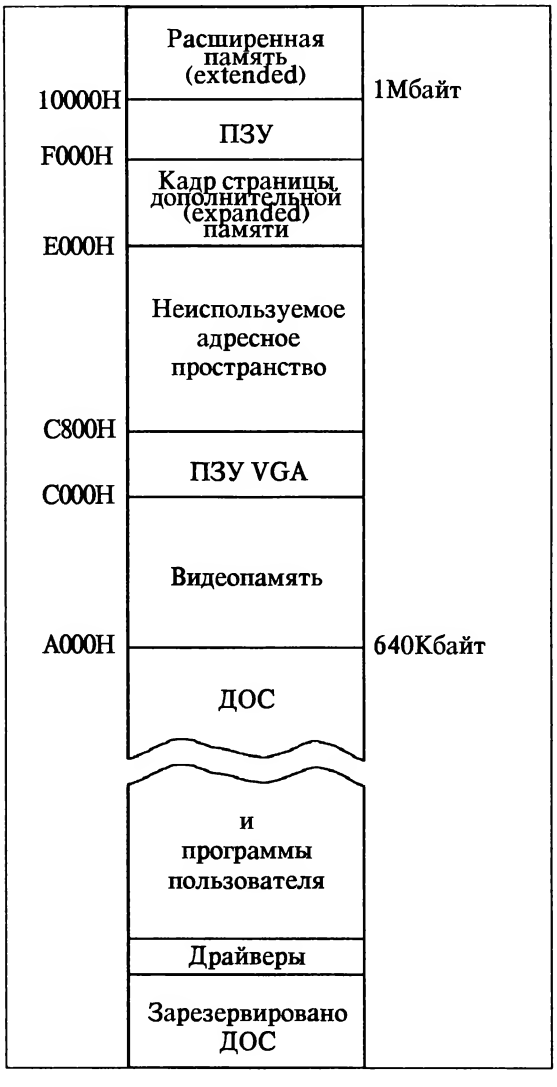
Увеличение объема оперативной памяти осуществляется двумя путями, первый из которых состоит в замене на системной плате микросхем памяти. Так, в руководстве некоторых ПЭВМ написано, что при замене 256-килобитных микросхем памяти на мегабитные объем оперативного запоминающего устройства (ОЗУ) может быть увеличен до двух или четырех мегабайт, в зависимости от количества используемых микросхем. Кроме этого, необходимо установить в новое положение несколько переключателей. В других ПЭВМ такие установки делаются программно, с помощью системной программы SetUp.

Второй путь состоит в покупке платы расширения памяти. Первый путь предпочтительнее, так как купить в нашей стране плату расширения не

так просто, и, что самое важное, неизвестно, будет ли она работать с Вашей машиной. Поэтому будьте внимательны при покупке ПЭВМ!

ДОС может работать только в стандартном режиме, поэтому драйверы, работающие с расширенной памятью (например, драйвер виртуальных дисков), должны обеспечивать переход в защищенный режим и обратно. К сожалению, фирма Intel не обеспечила простых путей переключения из защищенного в стандартный режим, и указанная операция может занимать до одной миллисекунды.

Рис.2. Распределение памяти в IBM PC



CMOS - память

Машины класса AT имеют так называемую CMOS (комплементарную металл - окисел - полупроводник) память. Эта память объемом 64 байта не входит в адресное пространство и используется для хранения конфигурации системы. Для сохранения информации при выключенном питании необходима батарейка, которая также служит для поддержки работы часов. Сразу отметим, что если Вы вытащите батарейку, или она придет в негодность, Вы сможете загрузить ПЭВМ только после реконфигурации системы. Программа реконфигурации (SetUp) обычно встроена в постоянное запоминающее устройство, но Вам необходимо на всякий случай обязательно списать все параметры текущей конфигурации, чтобы не оказаться неподготовленным в решающий момент.

CMOS-память, в частности, используется и для переключения из защищенного в стандартный режим в системах с МП 80286. Переключение осуществляется с помощью полного сброса состояния процессора. Перед сбросом необходимо сохранить в области данных BIOS адрес следующей команды и поместить в CMOS-память специальный завершающий код .

Сброс процессора переводит его в стандартный режим. Если в CMOS-памяти найден завершающий код, то работа процессора возобновляется в стандартном режиме с команды, адрес которой хранится в области данных BIOS, в противном случае осуществляется диагностика оборудования. Процедура перехода в стандартный режим может занимать до одной миллисекунды.

МП 80386 может работать с четырьмя гигабайтами памяти

МП 80386 является 32-битовым процессором, полностью совместимым с предыдущими моделями и обладающим более высокой производительностью. 80386 выпускается в двух модификациях: SX - для работы с 16-разрядной шиной и DX- с 32-разрядной. Использование 80386SX позволило разработчикам ПЭВМ с минимальными переделками выпустить сравнительно дешевые и производительные машины (тактовая частота 16 и 20 МГц). Системы на основе 80386DX более чем в полтора раза дороже, чем на основе 80386SX.

Каждая задача, выполняемая на МП 80386, может иметь в защищенном режиме до 16383 сегментов переменной длины общим объемом до четырех гигабайт [2]. МП 80386 имеет три различных адресных пространства: физическое, логическое и линейное.

Физические адреса - это реальные адреса, используемые аппаратурой для выборки и записи данных. Логический адрес состоит из указателя сегмента (16 бит) и смещения внутри сегмента (32 бита). Линейный адрес формируется добавлением внутрисегментного смещения к базовому адресу сегмента. Объем пространства логических адресов, отображаемый с помощью механизма трансляции адресов в физическое адресное пространство, достигает колоссальной цифры- 64 терабайта.

Преобразование логических адресов в физические производится в два приема: на первом этапе осуществляется трансляция логического адреса, состоящего из указателя и смещения, в линейный адрес, а затем трансляция линейного

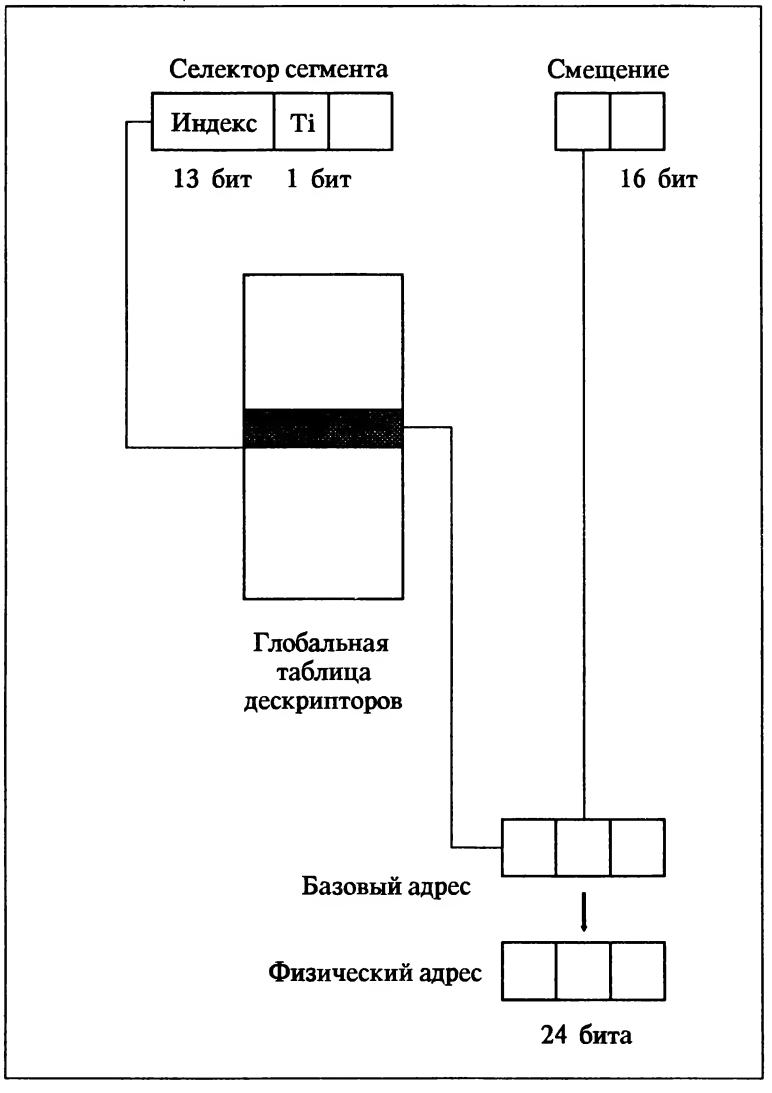


Рис.3. Механизм формирования адреса МП 80286 в защищенном режиме

адреса в физический. Второй этап необязателен и зависит от архитектуры операционной системы. Для нас существенно, что в защищенном режиме 80386 позволяет отображать логические адреса в любую область физического адресного пространства. Кроме этого, в МП 80386 упрощено переключение из защищенного режима в стандартный.

Режим виртуального процессора 8086

Еще одной особенностью МП 80386 является режим виртуального МП 8086, когда за счет аппаратных средств многозадачности, предоставляемых в защищенном режиме, обеспечивается одновременная работа нескольких операционных систем и прикладных программ на одном процессоре.

Для каждой пользовательской задачи создается виртуальная машина, полностью имитирующая МП 8086. Работу пользовательских задач обеспечивает системное программное обеспечение, обслуживающее внешний интерфейс, ввод/вывод, обработку прерываний. В качестве системной среды могут выступать такие надстройки над ДОС, как Windows-3 или DesqView. На рис. 4 приведена схема функционирования нескольких виртуальных машин в ДОС режиме виртуального процессора 8086.

Кэш-память

Увеличение быстродействия 80386 привело к тому, что время обращения к оперативной памяти значительно превысило время выполнения команд. Это обусловило необходимость введения иерархии запоминающих устройств. Кэш-память представляет собой быстродействующую буферную память, в которую команды и данные переносятся из оперативного запоминающего устройства. Принцип действия кэш основывается на том, что большинство команд и требуемых для их выполнения данных расположены "кучно", и основную часть времени процессор работает с кэш-памятью. Кэш-память гораздо дороже оперативной памяти, ее использование целесообразно только на высокопроизводительных машинах с тактовой частотой 33 МГц и более, обычно даже на самых дорогих ПЭВМ с МП 80486 объем кэш не превышает 64K. Обмен между ОЗУ и кэш-памятью полностью прозрачен для программиста и выполняется аппаратными средствами.

МП 80486

В МП 80486 меньше архитектурных новаций по сравнению с 80386. Верная своей традиции фирма Intel обеспечила программную совместимость с предшествующими разработками, начиная с 8086, сосредоточив внимание на производительности процессора. Дейст-

вительно, программа оценки производительности, запущенная на ПЭВМ с 80486, распознала его как 80386, работающий с тактовой частотой более 100 МГц. Большинство команд 80486 выполняются за меньшее число тактов по сравнению с 80386. Еще одним усовершенствованием является встроенный арифметический сопроцессор для выполнения вычислений с плавающей точкой, который в предшествующих моделях семейства МП 8086 представлял собой отдельную микросхему. ПЭВМ с 80486 выпускаются с тактовой частотой 25 и 33 МГц. Появились сообщения об экспериментальных системах, работающих в форсированном режиме на частоте 50 МГц, правда, для этого микросхему приходится охлаждать до минусовых температур. Появление серийных МП 80486 с тактовой частотой 50 МГц ожидается во втором полугодии 1991 года. В журнале Byte, N4 за 1991 год говорилось о работах фирмы Intel над 100 МГц версией МП 80486.

Дополнительная память

Есть еще один вид памяти, позволяющий "заглянуть" за 640 Кбайтный рубеж в стандартном режиме. Это так называемая дополнительная (expanded) память. Она также именуется EMS (expanded memory specification) или LIM по начальным буквам наименований трех фирм Lotus, Intel, Microsoft, выработавших стандарт дополнительной памяти.

Принцип действия дополнительной памяти базируется на аппаратном переключении адресов участков специально организованной оперативной памяти. В каждый момент времени пользователю доступен участок (кадр) дополнительной памяти размером 64 Кбайта, расположенный в пределах первого мегабайта. В частности, на рис. 2 "видимому" участку дополнительной памяти соответствуют 64 Кбайта, начиная с адреса E000h. Этот участок разделен на страницы по 16 Кбайт. Прикладная программа может осуществлять пересылки из "видимых" участков дополнительной памяти и в них. С помощью прерывания 67h осуществляется аппаратное переключение адресов, так что "видимыми" становятся другие страницы EMS. Объем дополнительной памяти может достигать 32 Мбайт.

Аппаратно реализованная дополнительная память при необходимости может быть использована как расширенная, но не наоборот. По сравнению с расширенной дополнительная память стоит несколько дороже.

Для работы с дополнительной памятью требуется программная поддержка в виде драйверов, например, EMM.SYS и QEMM.SYS, причем, они могут использовать для работы с EMS и расширенную память. Драйвер Above Disk позволяет, кроме того, моделировать дополнительную память в оперативной памяти и на твердом диске.

Основным достоинством дополнительной памяти является ее независимость от типа МП. Она с одинаковы 1 успехом может работать как на PC/XT, так и на высокопроизводительной рабочей станции с МП 80486. Кроме этого, дополнительная память является индустриальным стандартом, поддерживаемым производителями аппаратных и программных средств во всем мире.

К сожалению в дополнительной памяти можно хранить только данные, а не программный код, однако это ограничение, как будет показано во второй части статьи, иногда можно обойти.

Расширение стандарта дополнительной памяти, называемое EEMS, позволяет работать с участками памяти более 64 Кбайт. Последнее свойство используется системой DesqView для организации параллельной работы нескольких прикладных программ, для чего каждая прикладная программа грузится в EEMS, а переключение адресов позволяет практически мгновенно переместить ее в первые 640 Кбайт оперативной памяти и сделать активной.

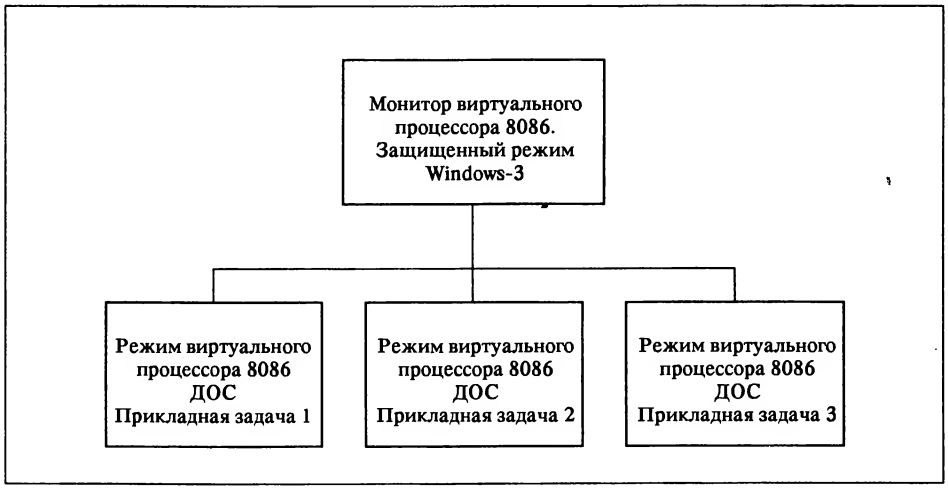
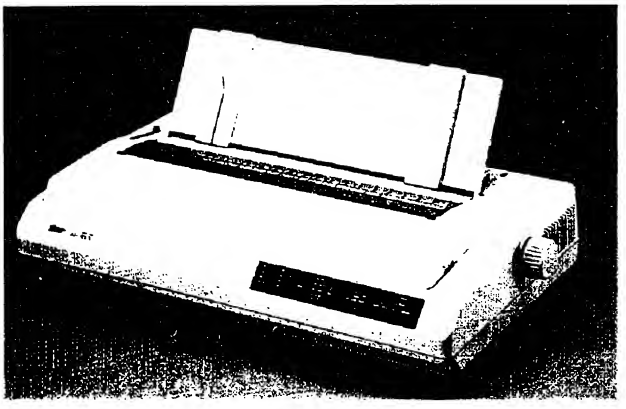


Рис. 4. Функционирование 80386 в режиме виртуального процессора 8086



Если Вы:

–занимаетесь издательской деятельностью;
– являетесь пользователем САПР;
– используете в работе математическое моделирование;
– имеете проблемы при разводке плат;
– являетесь разработчиком АРМ;
– занимаетесь комплексными научными исследованиями, –

То матричные принтеры фирмы "Star" избавят Вас от всех неудобств.

Широкие матричные принтеры фирмы "Star" модель XR–1500 –это:

– высокая скорость печати – 300 зн/сек;
– 8 шрифтов NLQ;
– большая емкость буфера – 32 К;
– возможность одновременного изготовления 4–х копий;
– наличие опций многоцветной печати;
– максимальная простота в управлении.

МНПВП "Элмос" предлагает принтеры модели XR–1500 фирмы "Star" и расходуемые материалы к ним за рубли.

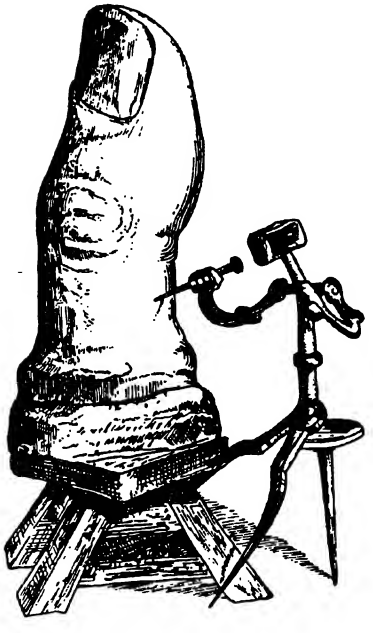
Обеспечивается гарантийное и послегарантийное обслуживание.

Телефон в Москве: 181–23–46

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ:

что это такое и для чего оно нужно

М. Черкашин



История первая или инкапсуляция

Вы программист на большом предприятии. Однажды Вас вызывает директор и говорит:
"Нужна информационно-справочная система. Срочно!"

Ну срочно, так срочно, что же теперь делать. Прикидываете: списки персонала цехов и отделов, финансовые документы, наряды, сметы... Словом, работы много. Начинаете писать программу (пример 1).

Пример 1

```
const
  Много = 10000;
type
  ВидПродукта = (ГАЙКИ,БОЛТЫ,ШУРУПЫ);

  УчетнаяКарточка = record
    Фамилия : string;
    Возраст : integer;
  end;

  Сотрудники = record
    ЧислоСотр : integer;
    СотрСписок : array [1..Много] of УчетнаяКарточка;
  end;

  Цех = record
    Персонал : Сотрудники;
    Продукт : ВидПродукта;
    Количество : integer;
    Цена : Integer;
  end;

  Отдел = record
    Персонал : Сотрудники;
    Начальник : string;
    Главбух : string;
  end;

function ПенсионерЛи (Список : Сотрудники; n : integer) : boolean;
begin
  with Список do
    ПенсионерЛи := (СотрСписок [n].Возраст > 55);
end;

procedure СписокСотрЦеха (Название : Цех);
begin
  with Название, Название.Персонал do
  begin
    for i:=1 to ЧислоСотр do
      writeln (СотрСписок [i].Фамилия);
  end;
end;

procedure СписокСотрОтдела (Название : Отдел);
begin
  with Название, Название.Персонал do
  begin
    for i:=1 to ЧислоСотр do
      writeln (СотрСписок [i].Фамилия);
  end;
end;

(* Программа выдает на печать списки сотрудников
  литейного цеха и отдела планирования. *)

Program;
var
  ОтделПланирования : Отдел;
  ЛитейныйЦех : Цех;
begin

(* Здесь присвоение значений переменным
  ОтделПланирования и ЛитейныйЦех.
  Эта часть программы опущена. *)

  СписокСотрОтдела (ОтделПланирования);
  СписокСотрЦеха (ЛитейныйЦех);
end.
```

Сначала все идет прекрасно, но потом программа усложняется, и Вы уже не можете сразу сказать, что делает та или иная процедура. Тогда для каждой процедуры Вы пишете, что она делает и для наглядности размещаете все эти записи в виде небольшой схемы программы (рис. 1).

В конце концов Вас сбивает с толку настойчивая просьба директора добавить к программе еще одну функцию. Программа теперь должна уметь печатать список всех пенсионеров цеха или отдела. А требования начальника продолжают расти.

Размер этой статьи не позволяет привести пример реального проекта. Впрочем, вряд ли в какой-нибудь книге Вам удастся найти такой. Но большие проекты все же существуют. Может быть, Вам самому приходилось сталкиваться с аналогичными проектами, или Вы знаете людей, которым выпала такая участь.

В программе, состоящей из тысяч, а то и десятков тысяч строк, разбираться, как показывает практика, значительно сложнее, чем в программе в несколько сот строк. Увы, это так. И в аналогичной программе такого размера разобраться, какая процедура работает с какими данными, уже представляет значительную проблему.

Итак, что же делать? Давайте внимательно еще раз посмотрим на схему программы. Рядом с каждым квадратиком, представляющим ту или иную структуру данных, выписаны все процедуры, которые с этой

структурой данных работают. И это совершенно естественно - просто мы для себя отвечаем на вопрос : "А что мы можем делать с этой структурой данных?". Но когда мы начинаем писать программу, нам приходится все это держать в голове или постоянно сверяться с табличкой. Таким образом, соответствие структуры программы написанной табличке находится целиком на совести программиста. А Вам не хочется немножко разгрузить себя и нагрузить машину? (Девиз фирмы IBM - "Машина должна работать, а человек думать").

Скажите, что есть такой объект ЗАВОД, что он состоит из подразделений, что ПОДРАЗДЕЛЕНИЕ - это либо ЦЕХ либо ОТДЕЛ, что для любого ПОДРАЗДЕЛЕНИЯ можно напечатать список сотрудников, а для любого ЦЕХА можно выдать ФИНАНСОВЫЙ ОБОРОТ и т.д. Наконец, сказав все это, добавьте, что можно выдать список пенсионеров любого подразделения. Ведь именно это Вам хочется сказать, не правда ли?

Назовем теперь объектом совокупность данных и процедур, с которыми эти данные работают, а тип, содержащий эти объекты - классом. Так, в нашем примере объектом будет отдел вместе с процедурами СПИСОКСОТРОТДЕЛА и РУКОВОДСТВООТДЕЛА, а все отделы, вместе взятые, образуют класс. Такой взгляд на вещи не является неестественным или даже чем-то новым. Например, едва ли мы будем звать автомобилем всякий предмет с корпусом, мотором и четырьмя колесами: автомобиль еще должен двигаться и перевозить людей, в противном случае это только его макет.

Две процедуры (см. рис.1), названные СПИСОКСОТРЦЕХА и СПИСОКСОТРОТДЕЛА, на самом деле делают одно и то же. Более того, их тексты дословно совпадают. Если бы мы ограничились объединением структур данных и процедур, нам так бы и пришлось писать по нескольку процедур для одних и тех же действий. Если Вам кажется, что не стоит овчинка выделки, что ради пары процедур не стоит и возиться, то не делайте поспешных выводов. Там, где могут быть две процедуры, может быть и больше, а писать одинаковые программы для десятка разных типов подразделений, согласитесь, не сахар. Для борьбы с подобными казусами придуман механизм, называемый наследованием.

Пусть ПОДРАЗДЕЛЕНИЕ - это объект, содержащий список сотрудников, а также процедуру печати этого списка. Пусть ЦЕХ - это такое подразделение, которое производит определенный товар в определенном количестве и за определенную цену. Тогда мы можем напечатать список сотрудников цеха уже потому, что цех сам по себе является подразделением. Говоря в терминах языка программирования, класс ЦЕХ является потомком класса ПОДРАЗДЕЛЕНИЕ, поэтому наследует процедуру печати списка сотрудников.

Отметим, что объектов типа ПОДРАЗДЕЛЕНИЕ не существует, как не существует и "просто подразделения" (любое подразделение будет либо цехом, либо отделом). Такие классы называются абстрактными классами.

Перепишем теперь программу, используя понятия объектно-ориентированного программирования (пример 2).

Примечание. В данном и последующих примерах используется близкий к Паскалю псевдокод, содержащий несколько объектно-ориентированных конструкций.

Эта программа выполняет ту

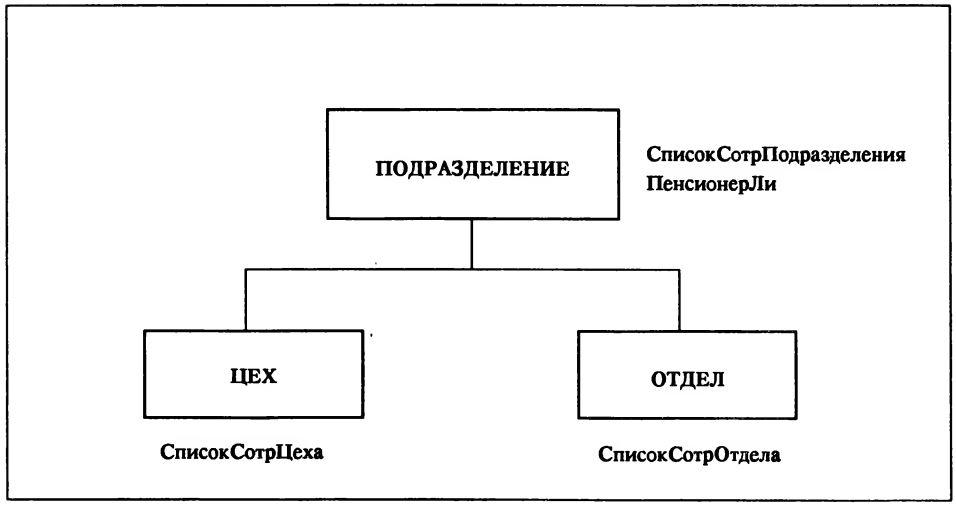
Пример 2

```
const Много = 10000;
type

  ВидПродукта = (ГАЙКИ, БОЛТЫ, ШУРУПЫ);


  УчетнаяКарточка = object
    Фамилия : string;
    Возраст : integer;
    function ПенсионерЛи : boolean;
    procedure ПечатьФамилии;
  end;

  Подразделение = object
    ЧислоСотр : integer;
    СотрСписок : array [1..Много] of УчетнаяКарточка;
    procedure СписокСотр;
  end;

  Цех = object (Подразделение)
    Продукт : ВидПродукта;
    Количество : integer;
    Цена : integer;
  end;


  Отдел = object (Подразделение)
    Начальник : string;
    Главбух : string;
  end;


function УчетнаяКарточка.ПенсионерЛи;
begin
  ПенсионерЛи := (Возраст > 55);
end;
```

```
procedure УчетнаяКарточка.ПечатьФамилии;
begin
  writeln (Фамилия);
end;


procedure Подразделение.СписокСотр;
var
  i : integer;
begin
  for i:=1 to ЧислоСотр do
    СотрСписок[i].ПечатьФамилии;
end;


(* Программа выдает на печать списки сотрудников ли-
  тейного цеха и отдела планирования *)
```

```
Program;
var
  ОтделПланирования : Отдел;
  ЛитейныйЦех : Цех;
begin


(* Здесь присваиваются значения переменным
  ОтделПланирования и ЛитейныйЦех.
  Эта часть программы опущена. *)

  ОтделПланирования.СписокСотр;

  ЛитейныйЦех.СписокСотр;

end.
```

Рис. 1



же задачу, что и ранее приведенная - является информационно-справочной системой предприятия. Однако по сравнению с первой программой ее структура стала намного яснее. Отметим тот факт, что в теле объекта объявляются только заголовки процедур, но не их тела. А при каждом описании тела процедуры указано имя процедуры и имя объекта, к которому она относится. Обратим теперь внимание на имя в скобках после ключевого слова object. Так указан факт, что описанный тип является потомком класса с указанным в скобках именем. Например, классы ЦЕХ и ОТДЕЛ объявляются потомками класса ПОДРАЗДЕЛЕНИЕ. И, наконец, внутри тела процедуры, относящегося к объекту, не указывается явно имя этого объекта. Поскольку программа знает, с каким объектом работает процедура, это становится излишним.

Пусть теперь нам надо добавить к этой программе процедуру печати списка всех пенсионеров какого-нибудь подразделения (т.е. цеха или отдела). Если бы мы использовали традиционный язык программирования , то нам бы пришлось писать две процедуры для двух типов подразделений, и, таким образом, вносить исправления в два места сразу. Используя же объекты, достаточно добавить к объекту ПОДРАЗДЕЛЕНИЕ одну процедуру и применить ее (в силу наследования), как для цеха, так и для отдела.

Итак, необходимо ввести в текст следующие добавления (пример 3).

Обратим внимание на реализацию процедуры ПЕНСИОНЕРЫПОДРАЗДЕЛЕНИЯ (объект ПОДРАЗДЕЛЕНИЕ). Там используется конструкция with. С помощью этой конструкции мы сообщаем машине, что мы будем работать с объектом СОТРСПИСОК[i]. Поэтому мы можем использовать процедуру ПЕЧАТЬФАМИЛИИ и функцию ПЕНСИОНЕРЛИ, принадлежащие этому объекту, без явного упоминания имени объекта. Таким образом, конструкция with используется с объектами подобно тому, как она используется с записями.

Более того, если выясняется,

что для какого-то особого подразделения список сотрудников выводится совсем по другому, то и это не страшно. Достаточно, не меняя ничего в объекте ПОДРАЗДЕЛЕНИЕ, написать так, как в примере 4, и процедура СПИСОКСОТР бу-

Пример 3

```
Подразделение = object
  ЧислоСотр : Integer;
  СотрСписок : array [1..Много] of УчетнаяКарточка;
  procedure ПенсионерыПодразделения;
  procedure СписокСотр;
end;

procedure Подразделение.ПенсионерыПодразделения;
var
  i : Integer;
begin
  for i:=1 to ЧислоСотр do
    with СотрСписок [i] do
      If ПенсионерЛи then
        ПечатьФамилии;
end;

(* Все остальное - как в примере 2. *)
```

Пример 4

```
НовоеПодразделение = Object (Подразделение)
  procedure СписокСотр;
end;

procedure НовоеПодразделение.СписокСотр;
begin

(* Здесь - любые действия, необходимые для того,
  чтобы распечатать список сотрудников. *)

end;
```

дет для этого класса аккуратно заменена на новую. При этом ни на других классах, ни на других процедурах этого класса данная замена не скажется.

История вторая или виртуальные процедуры

Немного времени спустя, к Вам опять приходит директор и дает новое задание. Ваша программа должна выдавать сводку подразделения. Постановка задачи.

1. Необходимо узнать имя должностного лица и цель запроса сводки. Напечатать это в правом верхнем углу листа бумаги перед заголовком. Далее напечатать заголовок. Выполнить действия, определяемые конкретным типом подразделения. Напечатать строчку "Программа работу закончила". Образец сводки прилагается.

2. Действия, специфичные для цеха: напечатать производимый товар, стоимость товара и объем его производства.

3. Действия, специфичные для отдела: напечатать фамилию главного бухгалтера.

4. Кроме того, через четыре месяца появится новый тип подразделения - производственный кооператив. Действия, специфичные для него, пока не известны, но, когда производственные кооперативы появятся, эти действия должны быть реализованы быстро.

Вообще говоря, процедура выдачи сводки подразделения различна в зависимости от того, сводку какого подразделения мы собираемся выдавать. Однако для значительной части описания безразлично, сводка цеха или отдела сейчас выдается. Это описание предполагает, что к моменту выполнения мы будем знать, что делать конкретно с текущим подразделением. Можем ли мы сказать машине: "Вот здесь вызови процедуру ДЕЙСТВИЯ-СПЕЦИФИЧ. Что это такое для каждого объекта, я пока не знаю, но перед тем, как тебе это придется выполнять, я это тебе сообщу." ? В традиционных языках не можем. А в объектно-ориентированых - вполн-

не. Процедуры описываются как обычно, только заголовок процедуры, подробное описание которой отложено до лучших времен, снабжается описателем virtual (в некоторых языках такие процедуры так и называются отложенными).

Пример 5

```
type

  ВидПродукта = (ГАЙКИ, БОЛТЫ, ШУРУПЫ);

  Подразделение = object
    ЧислоСотр : integer;
    СотрСписок : array [1..Много] of УчетнаяКарточка;
    procedure СписокСотр;
    procedure Сводка;
    procedure ДействияСпецифич; virtual;
  end;

  Цех = object (Подразделение)
    Продукт : ВидПродукта;
    Количество : integer;
    Цена : integer;
    procedure ДействияСпецифич; virtual;
  end;

  Отдел = object (Подразделение)
    Начальник : string;
    Главбух : string;
    procedure ДействияСпецифич; virtual;
  end;


procedure Подразделение.Сводка;
var
  КтоЗапросил, Цель : integer;
begin
  writeln('Лицо, запросившее сводку :');
  readln(КтоЗапросил);
  writeln('С целью :');
  readln(Цель);
  writeln('По запросу ',КтоЗапросил);
  writeln('Цель :', Цель);
  writeln('Сводка По Подразделению');
  ДействияСпецифич;
  writeln('Программа работу закончила');
end;


procedure Цех.ДействияСпецифич;
begin
  writeln (Продукт);
  writeln (Количество);
  writeln (Цена);
end;


procedure Отдел.ДействияСпецифич;
begin
  write ('Главный бухгалтер : ');
  writeln (Главбух);
end;
(* Все остальные процедуры в
  предыдущих примерах *)
```

Выше дана реализация процедуры СВОДКА (пример 5).

Для того чтобы добавить программу выдачи сводки для производственного кооператива, достаточно определить еще один класс ПРОИЗВОДСТВЕННЫЙКООПЕРАТИВ и для него процедуру ДЕЙСТВИЯСПЕЦИФИЧ. И все произойдет действительно быстро - не придется даже перекомпилировать то, что Вы раньше написали.

Подведем итоги. В отношении того, что считать объектно-ориентированным программированием, у разных специалистов существуют разногласия. Тем не менее обычно выделяются следующие три понятия: инкапсуляция, наследование и полиморфизм.

Инкапсуляция - это объединение процедур вместе со структурами данных, с помощью которого детали реализации тех или иных операций оказываются скрытыми от пользователя. Способность конкретизировать объект, т.е. позволять более узким классам объектов перенимать свойства от более широких (подобно тому, как класс ЦЕХ перенимает часть процедур от более общего класса ПОДРАЗДЕЛЕНИЕ) называется *наследованием*. Говоря о наследовании, мы неизбежно приходим к мысли, что для разных классов одни и те же (как по смыслу, так и по названию) операции должны осуществлять несколько разные действия. Такое принятие одной и той же операцией различных форм называется *полиморфизмом*. Так, процедура СПИСОКСОТРУДНИКОВ может означать печать списка сотрудников цеха или подразделения, а чего именно, иногда нельзя определить вплоть до момента выполнения.

Все основные понятия объектно-ориентированного программирования рассмотрены. Остается один вопрос : "А как же это все применять?" К программированию в терминах объектов, как впрочем и ко всякой другой новой идее, нужно сначала привыкнуть. Сейчас, наверное, самое луч-

шее - это посмотреть, как реально используются эти понятия в самых разных задачах.

Пример 1. Двунаправленные кольцевые списки

В программировании часто используется структура данных, называемая списком. Двунаправленный список - это такая структура данных, в которой каждый элемент "знает" местоположение двух соседних элементов - предыдущего и последующего. Элементы списка расположены в определенном порядке и поэтому обычно содержат первый и последний элементы. Однако вспомним выражение "круговая порука". Оно возникло тогда, когда крестьяне ставили свои подписи по кругу, чтобы никто не был первым - а нет человека, первым поставившего подпись, нет и зачинщиков. Так можно организовать и список. Упорядоченный подобным образом список называется кольцевым. Элементы в нем расположены по кругу (пример 6).

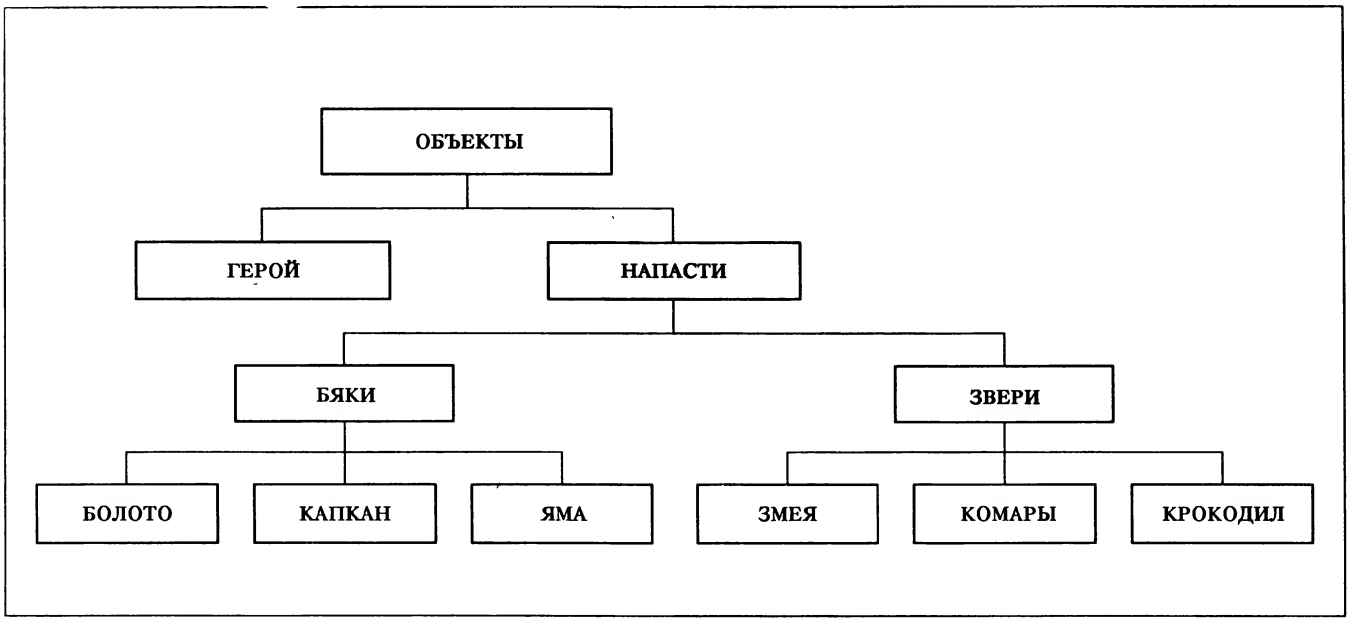
Таким образом, кольцевой список напоминает ожерелье. Держа в каждый момент времени только одну бусину ожерелья и перебирая бусины одну за другой, мы можем добраться до любой бусины. Конечно, добираться до определенного элемента списка довольно сложно.

Но зато легко можно находить последующий и предыдущий элементы, а также вставлять элементы в список и удалять их из списка. Эти действия не зависят от конкретной природы элементов, и поэтому очень хотелось бы написать для них процедуры, которые можно было бы использовать для различных типов элементов

Пример 6

```
type


  Элемент = object
    procedure ПечатьЭлемента; virtual;
  end;


  Список = object
    ТекущийЭлемент : ^Звено;
    ПервыйЭлемент : ^Звено;
    ПоследнийЭлемент : ^Звено;
    procedure СледующийЭлемент;
    procedure ПредыдущийЭлемент;
    procedure УдалитьЭлемент;
    procedure ВставитьЭлемент (Новый : ^Элемент);
    procedure НовыйСписок
    procedure ПечатьСписка;
  end;


  Звено = object
    ЭлементСписка : ^Элемент;
    Вперед, Назад : ^Звено;
  end;


procedure Список.НовыйСписок;
begin
  ТекущийЭлемент := nil;
end;


procedure Список.ПечатьСписка;
var
  Элем:^Звено;
begin
  write ('(');
  Элем := ПервыйЭлемент^.Вперед;

  if ТекущийЭлемент <> nil then
  begin
    Элем := ТекущийЭлемент;
    with Элем do
      repeat
        ПечатьЭлемента;
        СледующийЭлемент;
      until Элем = ТекущийЭлемент;
  end;
  writeln (')');
end;


procedure Список.СледующийЭлемент;
begin
  if ТекущийЭлемент <> nil then
    ТекущийЭлемент := ТекущийЭлемент^.Вперед;
end;


procedure Список.ПредыдущийЭлемент;
begin
  if ТекущийЭлемент <> nil then
    ТекущийЭлемент := ТекущийЭлемент^.Назад;
end;'


procedure Список.УдалитьЭлемент;
var
  След, Пред : ^Звено;
begin
  if ТекущийЭлемент = nil
    then writeln ('Попытка удалить элемент из пустого списка')
    else if ТекущийЭлемент.Вперед = ТекущийЭлемент
      then ТекущийЭлемент = nil
      else begin
        След :=ТекущийЭлемент^.Вперед;
        Пред := ТекущийЭлемент^.Назад;
        Пред^.Вперед := След;
        След^.Назад := Пред;
      end;
end;
```

Рис. 2




```
procedure Список.ВставитьЭлемент;
var
  НовоеЗвено:^Звено;
begin

  new (НовоеЗвено);

  if ТекущийЭлемент = nil then begin
    (* Вставить элемент в пустой список *)
    ТекущийЭлемент := НовоеЗвено;
    НовоеЗвено^.Вперед := ТекущийЭлемент;
    НовоеЗвено^.Назад := ТекущийЭлемент;
  end
  else begin
    (* Список не пуст *)
    След :=ТекущийЭлемент^.Вперед;
    ТекущийЭлемент^.Вперед := НовоеЗвено;
    НовоеЗвено^.Вперед := След;
    След^.Назад := НовоеЗвено;
    НовоеЗвено^.Назад := ТекущийЭлемент;
  end;

  НовоеЗвено^.ЭлементСписка :=Новый;

end:
```

списка. Однако попробуйте это сделать без использования объектов.

Обратимся к реализации объекта СПИСОК. Для того чтобы вставить новый элемент в список, процедуре ВСТАВИТЬЭЛЕМЕНТ необходимо только переставить ссылки двух элементов на новый элемент и установить ссылки самого нового элемента. Подобно этому удаление элемента осуществляется перестановкой ссылок.

Заметим, что до сих пор в программе нет ни малейшего намека на то, какого типа элементы этот список содержит. Теперь, когда нам необходимо создать список целых, достаточно написать пример 7.

▼ Пример 7

```
type
  ЦелыйЭлемент = Object (Элемент)
    ЦелоеЧисло : Integer;
    procedure ПечатьЭлемента; virtual;
  end;

(* Функция НовЭлем не входит в состав ни одного
   объекта. Ее задача - построить объект типа
   ЦелыйЭлемент, содержащий число, заданное
   параметром i. Используется здесь для упрощения
   задания элементов списка целых. *)
```

```
function НовЭлем (i : integer) : ^ЦелыйЭлемент;
var
  Новый : ЦелыйЭлемент;
begin
  new (Новый);
  Новый^.ЦелоеЧисло := i;
  НовЭлем := Новый;
end;


procedure ЦелыйЭлемент.ПечатьЭлемента;
begin
  write (ЦелоеЧисло);
end;

(* Программа, использующая списки *)

Program Списки;
var
  СписокЦелых : Список;
begin
  with СписокЦелых do
  begin
    НовыйСписок;
    ВставитьЭлемент (НовЭлем (5));
    ВставитьЭлемент (НовЭлем (4));
    ВставитьЭлемент (НовЭлем (3));
    ВставитьЭлемент (НовЭлем (2));
    ВставитьЭлемент (НовЭлем (1));
    ПечатьСписка;
    (* Будет напечатано (1 2 3 4 5) *)

    СледующийЭлемент;
    УдалитьЭлемент;
    СледующийЭлемент;
    УдалитьЭлемент;
    ПечатьСписка;
    (* Будет напечатано (1 3 5) *)
  end;
end.
```

Здесь мы описали объект ЦЕЛЫЙ ЭЛЕМЕНТ, процедуры печати целого элемента - и со списками целых уже можно работать. Без процедуры НОВЭЛЕМ можно обойтись, она используется только для удобства , чтобы нам было легче создавать элементы целого типа.

Создание механизма работы со списками - пример использования инкапсуляции. В самом деле, как только такой механизм создан, можно уже забыть о том, как он был реализован. Просто достаточно "сказать" машине :"А создай-ка ты мне список целых". Инкапсуляция здесь - цель, с которой используются объекты. Этой цели служат как наследование (возможность сказать : "ЦЕЛЫЙЭЛЕМЕНТ - это такой ЭЛЕМЕНТ"), так и полиморфизм (использование одних и тех же процедур вставки, удаления и т.д. для списков с различными типами элементов).

Пример 2."Как пишутся игрушки"

Пусть в Вашей компьютерной игре за героем охотятся ЗВЕРИ и его преследуют БЯКИ. Хотя с точки зрения игрока герой просто удирает от всех этих напастей и все происходит одновременно, на самом деле происходит следующее. Сначала вводится с клавиатуры символ (например стрелка), в зависимости от него как-то перемещается герой, затем БЯКИ и ЗВЕРИ (единообразно) определяют направление на героя и затем движутся по направлению к нему. Способ этого движения у БЯК свой, а у ЗВЕРЕЙ свой. А именно: БЯКИ возникают у героя на пути, а ЗВЕРИ гонятся за ним.

Зверь - это крокодил, змея или комары. Что делает зверь, столкнувшись с героем, зависит от него (комары уменьшают вдвое жизненную силу героя, крокодилы перекусывают его пополам). Бяка - это болото, капкан или яма. Что происходит с героем, попавшим в бяку, зависит от ее типа. Чтобы читатель не запутался, приведем небольшую диаграмму "Что есть что" (рис. 2). Диаграмма "Что ест что" очень проста - все ест главного героя - и поэтому не приводится.

А теперь приведем набросок программы, эту игрушку реализуюшей (пример 8).

Теперь, если выясняется, что герой попался или он был сьеден зверем, то достаточно написать ГЕРОЙПОПАЛСЯ, и

Пример 8

```
type
  Объект = object
    X,Y : integer; (* Координаты *)
    НаДисплее : Изображение;
    procedure НарисоватьИзображение (X,Y : integer);
    procedure СтеретьИзображение;
    procedure ПереместитьИзображение(НовыйX, НовыйY);
  end;

Герой = object (Объект)
end;


Напасть = object (Объект)
  НаправлениеНаГероя : integer;
  procedure ОпределитьНаправлениеНаГероя;
  procedure ГеройПопалсяЛи;
```

```
    procedure ГеройПопался; virtual;
  end;


  Бяка = object (Напасть)
    procedure ВнезапноПоявиться;
    procedure ВозможноеПеревоплощение;
  end;


  Болото = object (Бяка)
    procedure ГеройПопался;
  end;


  КАПКАН = object (Бяка)
    procedure ГеройПопался;
  end;


  ЯМА = object (Бяка)
    procedure ГеройПопался;
  end;


  Зверь = object (Напасть)
    procedure БежатьЗаГероем;
  end;


  Змея = object (Зверь)
    procedure ГеройПопался;
  end;


  Комары = object (Зверь)
    procedure ГеройПопался;
  end;


  Крокодил = object (Зверь)
    procedure ГеройПопался;
  end;


var
  Счет, ЧислоЖизней, ЖизненнаяСила : integer;
  Игрок : Герой;

procedure Напасть.ГеройПопалсяЛи;
begin
  if Игрок.X = X and Игрок.Y = Y then
    ГеройПопался;
end;


procedure Бяка.ВозможноеПеревоплощение;
  if СлучайноеЧисло = 0 then begin
    СтеретьИзображение;
    ОпределитьНаправлениеНаГероя;
    ВнезапноПоявиться;
end;

procedure Зверь.БежатьЗаГероем;
var
  Направление : integer;
  НовыйX, НовыйY : integer;
begin
  ОпределитьНаправлениеНаГероя (Направление);
(* Определить с помощью текущих координат
  и полученного направления новые координаты
  НовыйX и НовыйY *)
  ПереместитьИзображение (НовыйX, НовыйY);
end;


procedure Комары.ГеройПопался;
begin
  ЖизненнаяСила := ЖизненнаяСила div 2;
end;


procedure Крокодил.ГеройПопался;
begin
  ЧислоЖизней := ЧислоЖизней - 1;
end;

(* В этой программе используются объекты
  и процедуры,определенные выше. *)


program Игра;
var
  Бяка1, Бяка2, Бяка3 : Бяка;
  Зверь1, Зверь2 :Зверь;
begin
  Счет := 0;
  ЧислоЖизней := 3;
  ЖизненнаяСила := 100;

  while ЧислоЖизней 0 do begin
    Читать стрелку;
    Считать новые координаты
    НовыйX и НовыйY;
    СчитатьИОпределитьНовыеКоординаты;
    Игрок:ПереместитьИзображение (НовыйX,НовыйY);

    Бяка1.ВозможноеПеревоплощение;
    Бяка2.ВозможноеПеревоплощение;
    Бяка3.ВозможноеПеревоплощение;
    Зверь1.БежатьЗаГероем;
    Зверь2.БежатьЗаГероем;

    Бяка1.ГеройПопалсяЛи;
    Бяка2.ГеройПопалсяЛи;
    Бяка3.ГеройПопалсяЛи;
    Зверь1.ГеройПопалсяЛи;
    Зверь2.ГеройПопалсяЛи;
  end;
end.
```

все необходимые действия будут выполнены.

Пример 3. Ввод-вывод

Программистский фольклор говорит: "Во всякой программе есть две большие части - ввод-вывод и все остальное". Действительно, в реальной программе этот самый ввод-вывод может поглощать львиную долю усилий программиста. Выводить приходится не только разные структуры данных (различные виды таблиц, бланков, меню), но и на различные устройства (принтер, экран монитора, графопостроитель). Поэтому, если Вы сейчас пишете "все остальное", то о вводе-выводе лучше не думать. И если в вашем расположении только традиционные языки, то приходится "выкручиваться", подчас не самым удобным образом. Если же у Вас объектно-ориентированный язык, то Вы можете спокойно написать пример 9.

Пример 9

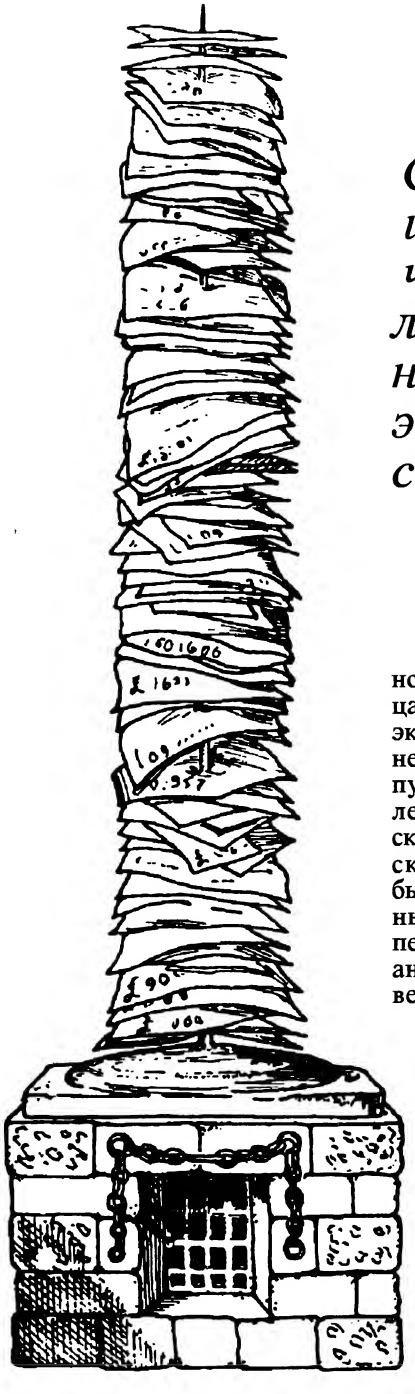
```
type

  Устройство = object
    procedure ПечатьТаблицы; virtual;
    procedure ПечатьФормы; virtual;
  end;


  Принтер = object (Устройство)
    procedure ПечатьТаблицы; virtual;
    procedure ПечатьФормы; virtual;
  end;

  Экран = object (Устройство)
    procedure ПечатьТаблицы ; virtual;
    procedure ПечатьФормы ; virtual;
  end;


Program;
var
  КакоеНибудьУстройство : Устройство;
begin
.
  КакоеНибудьУстройство.ПечатьТаблицы;
.
end.
```

В данном примере использование объектов не сокращает количество строк программы, которое Вам необходимо написать. Однако объекты здесь позволяют Вам работать с понятием устройства как с единым целым. Выбрав необходимое устройство в одном месте программы, в дальнейшем Вы уже можете не обращать внимания на то, с каким именно устройством Вы сейчас работаете.

И даже, если потом у Вас появится с десяток новых устройств, Вы можете просто добавить по новому объекту для каждого из них и в каждый из объектов - необходимые процедуры ввода-вывода. После этого все, что Вам приходится менять - это та часть программы, в которой Вы выбираете, на какое устройство выводить. Право же, совсем немного.

Сколько экспертных систем уместится на кончике иглы ?

И. Исаев



Сейчас и представить трудно, что всего десять-пятнадцать лет назад огромного мира экспертных систем еще не существовало.

Сейчас и представить трудно, что всего десять-пятнадцать лет назад огромного мира экспертных систем (ЭС) еще не существовало. В то время из пучин Искусственного Интеллекта поднимались лишь несколько островков вулканического происхождения. Это были чрезвычайно оригинальные и многообещающие экспертные системы: DENDRAL - анализ данных с целью вывода вероятных структур неизвестного химического соединения; MACSYMA - символьные математические преобразования; EXPERT - язык для построения экспертных систем; MYCIN - диагностика и лечение инфекционных заболеваний крови; TEIRESIAS - система, помогающая при создании больших баз знаний; HEARSAY-II - система понимания речи и некоторые другие. Работа экспертных систем вызывала восхищение у современников. Машина успешно решала такие задачи, с которыми раньше могли справиться лишь умудренные опытом доктора, почтенные специалисты-химики или исходившие не одну тысячу миль геологи. Например, система MYCIN давала консультации по диагностике и лечению инфекционных заболеваний крови. В 1982 году MYCIN содержала около 100 тысяч ассоциативных связей, которые составляли 85% всех знаний в этой области медицины. По мнению экспертов, MYCIN действовала на уровне врачей-специалистов, а в отдельных случаях и лучше.

Окончательное оформление всего, что связано с ЭС, в самостоятельную область искусственного интеллекта можно отнести к началу 80-х годов. Именно тогда появилось большое число теоретических исследований и практических результатов, посвященных проблематике ЭС.

Несколько слов о том, что же такое экспертная система. ЭС состоит из следующих основных частей:

-базы данных;
-базы знаний;
-машины вывода;
-блока, помогающего при диалоге человека и машины (интерфейс "человек-машина").

Кроме этого, в состав ЭС могут входить блок приобретения знаний и блок объяснения действий машины. На рис.1 показаны связи между главными блоками ЭС.

База данных содержит исходные и промежуточные данные, необходимые для решения текущей задачи.

В базе знаний хранятся факты из какой-либо предметной области (например, медицины или схемотехники), а также правила (есть более научный термин: эвристики), которые используются при решении задач.

Машина вывода формирует такую последовательность правил, которая приводит к решению задачи. Для этого машина вывода обращается и к базе данных, и к базе знаний.

Интерфейс "человек-машина" помогает пользователям работать с ЭС в ходе решения задач, приобретения знаний и обьяснения результатов работы.

Блок объяснения показывает, как система получила решение задачи и какие знания она при этом использовала.

Блок приобретения знаний помогает автоматизировать процесс передачи знаний от специалиста (эксперта) к ЭС.

Для большей наглядности работу ЭС можно сравнить с литейным процессом. Расплавленные огнедышащие данные (т.е. базу данных) "заливают" (работа машины вывода) в замысловато изогнутые формы (для этого используется база знаний). Можно привести другое сравнение. Работа ЭС очень напоминает повара, приготовляющего еду. В этом случае продукты - это данные, кулинарная книга - знания, повар - машина вывода, меню - интерфейс "человек-машина".

Необходимо отметить то обстоятельство, что знания специалиста (эксперта) являются неотъемлемой частью ЭС. Причем, в контексте ЭС наиболее важными считаются те знания, которые накоплены экспертом в результате своей профессиональной деятельности, и которые, как правило, отсутствуют в учебниках или справочниках по данной области. Именно индивидуальные знания помогают опытному эксперту (например, врачу при постановке диагноза) решать трудные неформализованные задачи.

Знания всегда стоили очень дорого. Передача знаний от специалиста к новичку всегда была длительным и сложным процессом. В этой связи очень соблазнительной выглядит перспектива иметь сказочный сундучок, который может беседовать с экспертом о тех или иных знаниях, а затем формализовать, уточнять и воспроизводить их. Конечно для этого в сундучке должна оказаться хорошая экспертная система, созданная добрым волшебником (наполовину ученым, наполовину программистом) - инженером по знаниям.

На сегодняшний день существуют сотни ЭС в десятках прикладных областей. Разработаны и успешно эксплуатируются программные средства для создания ЭС. Все эти продукты успешно продаются и приносят большой экономический эффект.

Остановимся чуть подробнее на применении ЭС. Очень вероятно, что при все еще небольшой распространенности экспертных систем и сведений о них, эта тема может напомнить "чистые" рассуждения средневековых богословов о количестве чертей, способных уместиться на острие иголки. Однако роль ЭС среди прочих программных продуктов настолько велика, что информация о высокоэффективных программах для специализированных, профессиональных предметных областей будет полезна самому широкому кругу читателей. Обратимся к фундаментальному труду [1], где приведена одна из самых удачных классификаций ЭС. Все многообразие применений ЭС условно разделено на нс-

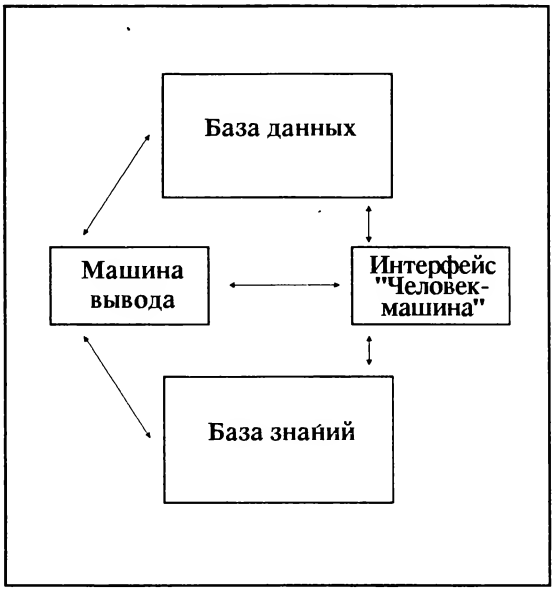


Рис. 1. Связи между главными блоками ЭС

сколько типов, описанных ниже:

- ИНТЕРПРЕТАЦИЯ: построение описания ситуаций по наблюдаемым данным;
- ПРОГНОЗ : вывод вероятных следствий из заданных ситуаций;
- ДИАГНОСТИКА : заключение о нарушениях в системе, исходя из наблюдений;
- ПРОЕКТИРОВАНИЕ : построение конфигурации объектов при ограничениях;
- ПЛАНИРОВАНИЕ : проектирование плана действий;
- МОНИТОРИНГ : сравнение наблюдений с критическими точками плана;
- ОТЛАДКА : выработка ре-

комендаций по устранению неисправностей;

- РЕМОНТ : выполнение плана применения выработанной рекомендации;
- ОБУЧЕНИЕ : диагностика, отлаживание и исправление поведения ученика;
- УПРАВЛЕНИЕ : интерпретация, прогноз, ремонт и мониторинг поведения системы.

Обратите внимание на постепенно возрастающую сложность типов применений ЭС, при которой каждый последующий тип может объединять несколько предыдущих. Кроме этого, данная классификация хорошо согласуется с этапами развития ЭС (от DENDRAL и MYCIN до таких монстров, как ЭС, разрабатываемые в рамках СОИ).

Рис. 2. Распределение ЭС по областям применения

Интерпретирующие системы выводят описание ситуаций из наблюдаемых данных. К этой категории относятся распознавание, понимание речи, анализ изображений, определение структуры химического соединения, интерпретация символов и многие виды анализа информации.

Прогнозирующие системы логически выводят вероятные следствия из заданных ситуаций. К этой категории относятся предсказание погоды, демографические прогнозы, предсказание дорожной обстановки, оценки будущего урожая и прогнозы в военной области.

Системы диагностики судят о нарушениях в работе системы по наблюдениям. К этой категории относится среди прочего диагностика в медицине, электронных схемах, механических устройствах и в системах программного обеспечения.

Проектирующие системы объединяют простые объекты в сложную конструкцию, в соответствии с заданными ограничениями. К таким задачам относятся синтез и компоновка электронных схем, проектирование зданий и составление бюджета.

Системы планирования проектируют планы действий. Сюда относятся автоматическое программирование и задачи планирования поведения робота, планирование при проектировании, планирование в военной области.

Системы мониторинга (предупредительные системы) сопоставляют результаты наблюдений за поведением системы с критическими точками плана. Существует множество систем мониторинга для атомных электростанций, воздушного движения, для задач, связанных с регулированием и управлением финансами.

Системы отладки дают рекомендации для устранения неисправностей. Существуют машинные системы отладки программ для вычислительных машин.

Системы, относящиеся к типу РЕМОНТ, выполняют план действий, следующий из выработанной рекомендации. Системы такого типа встречаются в автомобильной промышленности, в авиации и для обеспечения работы компьютеров.

Системы обучения диагностируют и отлаживают "поведение студента". В них используются подсистемы диагностики и отладки, специально ориентированные на студента.

Экспертная управляющая система обеспечивает гибкое управление всем поведением некоторой системы. К задачам, решаемым управляющими системами, относятся управление воздушным транспортом, управление деловой активностью, управление боем.

На рис. 2 показано распределение ЭС по областям применения. 100% (полный круг) соответствует 152 рассмотренным системам (данные 1987 года).

Приведем несколько впечатляющих примеров эффективности ЭС. Применение ЭС дает возможность повысить производительность труда при создании СБИС в 3-4 раза, ускорить поиск неисправностей в 5-7 раз, сократить в 8-10 раз затраты на профессиональную подготовку специалистов. В [2] можно найти следующие любопытные цифры, характеризующие современное состояние в сфере ЭС. Опрос 300 английских ведущих фирм показал, что в 1985 году 54% фирм использовали ЭС, 20% планировали их использование. В 1987 году

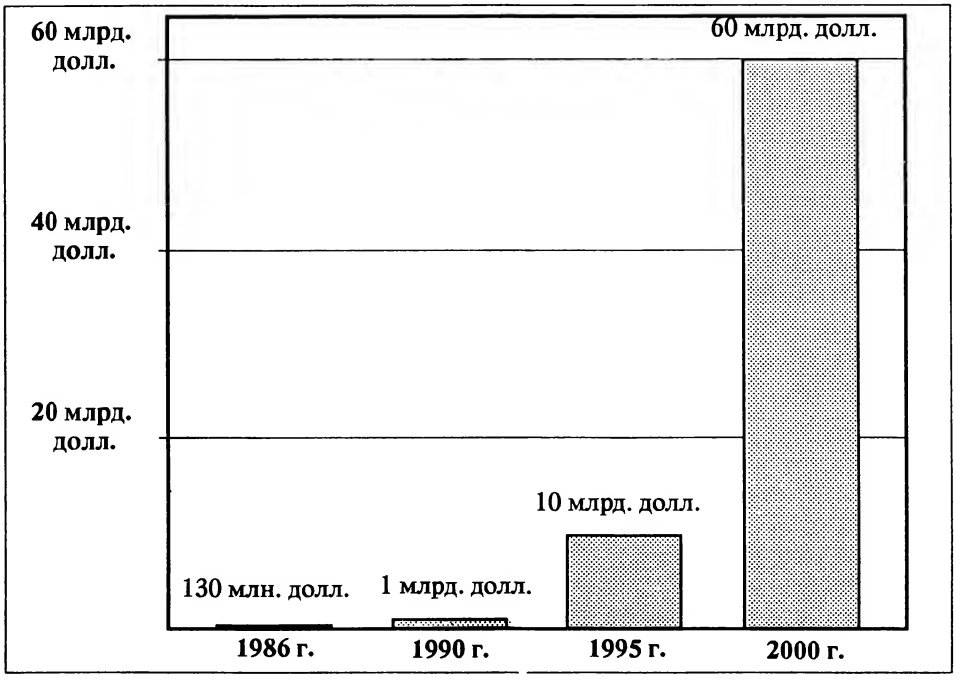


Рис. 3. Динамика рынка ЭС в США

практически все компьютерные фирмы США имели лаборатории и отделения по ЭС. На рис.3 показана динамика роста рынка ЭС в США. На горизонтальной оси отложены годы, на вертикальной оси - общая сумма (в миллиардах долларов) средств, связанных с разработкой, эксплуатацией и продажей ЭС.

Несмотря на значительную стоимость разработки, ЭС приносят существенный доход. Например, доход от ЭС XCON (помогает продавцам системы VAX фирмы DEC определять конфигурацию системы) составляет 25 млн. долларов в год при стоимости разработки около 3 млн. долларов.

Продолжая тему использования ЭС, предлагаем Вашему вниманию табл. 1-3, в которых собраны данные (ноябрь 1990 года), касающиеся причин разработки ЭС, основных областей применения, а также аппаратного обеспечения. 100% в таблицах соответствуют 89 американским фирмам, принявшим участие в опросе.

Несомненно будет совсем не лишним привести некоторые высказывания ученых, плодотворно занимающихся ЭС.

- " ЭС дает возможность любому, кто обладает каким-либо профессиональным опытом, составить схему, в общих чертах отражающую ход решения конкретной задачи."
  Б.Томпсон, У.Томпсон.
- " Любая ЭС - знания и данные, представленные в виде, удобном для машинной обработки, и организованные "дружелюбным" для использования непрограммирующими профессионалами образом."
  В.Ф. Хорошевский.
- " ЭС выглядит почти как карикатура на специалиста-человека, который узнает все больше о все меньшем."
  Р. Форсайт.
- " К ЭС можно даже обратиться с таким вызывающим вопросом, как:"Почему Вы не учли, что ...?" - на что программа объяснит, почему та или иная возможность была исключена из рассмотрения."
  А. Эндрю.
- " Перед тем как приступить к разработке ЭС, инженер по знаниям должен рассмотреть вопрос, следует ли разрабатывать ЭС для данного приложения."
  Э.В. Попов.
- " Создание конкретных ЭС в принципе не ограничено какими-либо явными факторами. Тем не менее, по мере того, как сложность и размеры ЭС возрастают, "камнем преткновения" на пути их дальнейшего развития становится персональный компьютер - в силу

Таблица 1. Аппаратное обеспечение ЭС

Аппаратное обеспечение ЭС	Количество фирм, использующих данную технику (%)
IBM PC	83
Macintosh	19
Microvax	9
Sun 386	9
HP 9000	6
DEC Workstation	5
Другое	13

ограниченности его собственных возможностей." Д. Мичи, А. Буланже, Р. Майклсен.

- " Наиболее значительным побочным продуктом работы над ЭС будет кодирование знаний. Человеческая культура обогащается при кодировании знаний, и всякая возможность автоматизации познания приведет к новым достижениям." Ф. Хейес-Рот, Д. Уотерман, Д. Ленат.
- " ЭС, наконец, перечеркнули старое высказывание: если нечто работает, то это уже не искусственный интеллект." Р. Форсайт.

И в заключение несколько слов о перспективах расширения областей использования ЭС. В ближайшем будущем можно ожидать широкого применения ЭС в экологии, управлении энергетическими ресурсами, психологии и компьютерных играх. ЭС будут лучше осознавать свои действия в ходе решения задач (к чему стоит стремиться и всем здравомыслящим людям). Такой напарник, как экспертная система, поможет Вам в любой ситуации (если в сети есть напряжение). Поэтому новое поколение выбирает экспертные системы!

Список литературы.
1. Построение ЭС./Под ред. Ф. Хейеса-Рота, Д. Уотермана, Д. Лената.-М.:Мир,1987.-С.24-26.
2. Справочник по искусственному интеллекту.Том 3./Под ред. Э.В. Попова.-М.:Радио и связь,1990.-С.261-262.

Наиболее известные экспертные системы

Экспертная система DENDRAL анализирует данные ядерного магнитного резонанса, масс-спектрографические и другие химические экспериментальные данные с целью вывода вероятных структур неизвестного химического соединения. Система DENDRAL при решении своих задач значительно превосходит возможности человеческого интеллекта.

MACSYMA - экспертная система для проведения символьных преобразований математических выражений, созданная в Массачусетском технологическом институте, США. Система MACSYMA превосходит возможности большинства специалистов. Система производит дифференцирование и интегрирование, и отлично справляется с задачей упрощения символьных математических преобразований.

Экспертная система R1 (современная версия называется XCON) помогает покупателю сформулировать требования к конфигурации вычислительной системы VAX на фирме DEC. Для ЭВМ VAX производится более 400 компонентов, и число вариантов систем столь велико, что задача определения конфигурации по силам лишь очень опытным программистам. R1 успешно конкурирует с людьми при составлении конфигурации VAX и приносит ежегодный доход в несколько миллионов долларов.

Области применения ЭС	Количество фирм, предполагающих иметь ЭС в данной области (%)
Производство продукции	44
Организация производства	29
Бухгалтерский учет	20
Финансовые операции	14
Торговля	14
Образование	9
Конструирование	6

Таблица 2. Области применения ЭС

Причины разработки ЭС	Количество фирм, заинтересованных в решении задач такого рода(%)
Повышение производительности труда	63
Улучшение уровня обслуживания	53
Повышение качества продукции	51
Повышение уровня диагностики	45
Сохранение знаний	44
Повышение качества обучения	31
Повышение технологического уровня	29
Помощь в конструкторских разработках	21
Другое	7

Таблица 3. Причины разработки ЭС

МП "Ником"

Принимает от промышленных предприятий, предприятий торговли, других организаций и частных лиц заявки на поставку, начиная с января 1992 года,

телефонного аппарата "Nicom 1".

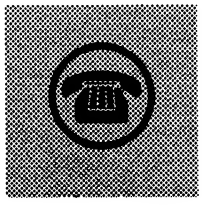
"Nicom 1" - это:

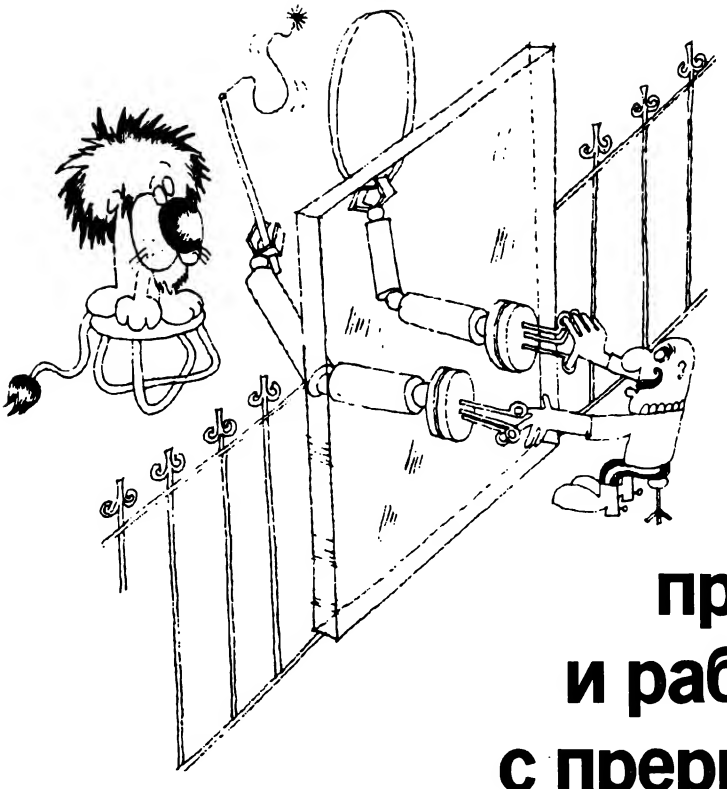
- эффектный плоский пластмассовый корпус разных цветов;
- мягкое нажатие клавиш;
- память последнего номера;
- мелодичный вызов;
- идеальное сопряжение с отечественной телефонной сетью.

Цена не выше 300 рублей

Оптовым покупателям гарантируется скидка

Наш адрес:
Москва, ул. М. Ульяновой, д. 16, строение 1.
Телефон: 235-99-45





М.Остринский

Создание резидентных программ и работа с прерываниями

Для чего нужны резидентные программы

Существуют программы, остающиеся в памяти компьютера после завершения своей работы. Эти программы носят название резидентных или TSR-программ (от английского словосочетания "Terminate and State Resident" - "завершиться и остаться в памяти").

Резидентные программы - это постоянно находящиеся в памяти программы помощи (POP UP HELP), калькуляторы, справочники и словари, программы копирования и качественной печати экрана, русификаторы, вирусы и многое другое.

Находясь в оперативной памяти, резидентная программа пассивна до наступления ключевого события - таким событием может быть прерывание.

При активизации резидентной программы не имеет значения, какая программа выполняется в этот момент, и нет необходимости завершать ее работу.

В дальнейшем мы будем называть программу, выполняющуюся в момент активизации резидентной программы, основным процессом, а саму резидентную программу - фоновым процессом.

Что такое прерывание

Прерыванием называются действия, производимые компьютером для выполнения системных и сервисных функций во время работы основного процесса.

Если основной процесс - просмотр текста в экранном редакторе, то примером прерывания является вызов резидентного калькулятора.

Процесс прерывания (в дальнейшем его будем называть просто прерыванием) проходит в три стадии :

1) приостановка выполнения основного процесса;

2) обработка прерывания;

3) возврат управления основному процессу.

Резидентная программа собственно и осуществляет обработку прерываний.

Здесь следует сделать уточнение. Программист пишет программу, логически анализирующую возникшую ситуацию - это обработка прерывания на высоком уровне. Для выполнения действий по управлению аппаратурой (ввод/вывод символа, чтение с жесткого диска и т.д.) используются процедуры базовой системы ввода/вывода BIOS (Base Input/Output System). Обращение к ним из программы

также называется прерыванием, более точно - сервисным прерыванием.

Отличием программ обработки прерываний от обычных подпрограмм, является то, что система перед их вызовом помещает в стек регистр слова состояния процессора (PSW) для его восстановления в конце прерывания, а управление всегда передается с помощью косвенной адресации.

Аппаратные и сервисные прерывания

Рассмотрим в качестве примера задачу контроля за функционированием химического реактора. Требуется автоматизировать принятие экстренных мер по сигналу с датчиков, контролирующих параметры процесса, в случае отклонения этих параметров от заданных.

Невозможно заранее предугадать, когда произойдет подобное событие. Компьютер в этот момент может выполнять любую программу, например, тестировать обслуживающий персонал на знание техники безопасности или управлять оптимальной загрузкой производственного оборудования. Когда поступает сигнал с датчиков о нештатной работе, выполнение основного процесса должно быть прервано.

Такие прерывания называются асинхронными или аппаратными. Они генерируются системой MS DOS при получении от оборудования сигнала о каком-либо событии.

Аппаратные прерывания нумеруются в порядке приоритета вызова от 0-го до 15-го для IBM PC/AT и от 0-го до 7-го для IBM PC/XT. Все эти прерывания, кроме 2-го (IRQ2), могут быть запрещены ("маскированы") путем сброса флага "I" в регистре слова состояния процессора (бит 7), и затем снова разрешены обратным действием. Не рекомендуется надолго (более 1/18 секунды реального времени) запрещать прерывания, так как это приведет к отставанию системных часов от абсолютного времени.

В отличие от аппаратных, сервисные прерывания происходят только если в программе есть явное указание произвести прерывание. Сервисные прерывания используются для реализации функций работы с внешними и внутренними устройствами, такими, как диски, экран дисплея, принтеры, мышь и т.д. Практически все сервисные прерывания могут выполнять большой набор функций для реализации различных операций работы с внешними устройствами.

Как вызвать сервисное прерывание

Каждое прерывание имеет уникальный порядковый номер. В руководствах по MS-DOS и BIOS для каждого прерывания описаны действия, выполняемые компьютером. Например, чтобы прочитать символ, вводимый с клавиатуры, необходимо осуществить прерывание с номером 16H и функцией 00H.

По команде

MOV AH,00H

номер функции загружается в регистр AH.

По команде

INT 16H

происходит прерывание, и символ, введенный с клавиатуры, помещается в регистр AL.

В таблице 1 приведены примеры аппаратных и сервисных прерываний для IBM PC/AT.

Некоторые из этих прерываний дополняют друг друга. При нажатии клавиши на клавиатуре, системой генерируется аппаратное прерывание INT 09H (IRQ1), при обработке которого сканкод нажатой клавиши помещается в буфер клавиатуры, а одна из функций сервисного прерывания INT 16H позволяет считать этот код из буфера клавиатуры и возвращает ASCII-код символа, соответствующего нажатой клавише.

Таблица 1. Примеры аппаратных и сервисных прерываний

Аппаратные прерывания			Сервисные прерывания	
N (hex)	Системное название	Назначение	N (hex)	Назначение
08	IRQ 0	Таймер	1A	Работа с часами
09	IRQ 1	Клавиатура	16	Чтение клавиши
0B	IRQ 3	COM2	14	COM I/O
0C	IRQ 4	COM1	10	Дисплейный I/O
0D	IRQ 5	LPT2	05	Печать экрана
0E	IRQ 6	Контроллер FD	13	Дисковый I/O
0F	IRQ 7	LPT1	17	Принтерный I/O

Как устанавливаются в системе резидентные программы обработки прерываний

В оперативной памяти компьютера первые 256*2 слов зарезервированы для хранения сегментных адресов программ обработки прерываний. Например, адрес программы обработки прерывания от клавиатуры INT 16H находится в ячейках 16H*2 (базовый адрес сегмента) и в 16H*2+1 (смещение). Зарезервированное поле адресов программ носит специальное название - область векторов прерываний. При возникновении события, вызывающего прерывание, система читает соответствующий событию вектор прерывания и вызывает программу обработки прерывания, помещенную по адресу, на который указывает этот вектор. Процесс обработки прерывания считается завершенным, когда программа обработки прерывания засылает в порт с адресом 20H микросхемы контроллера прерываний код 20H.

Разработчики аппаратуры персонального компьютера придерживались идеологии "открытой архитектуры". К компьютеру можно подключать множество периферийных устройств, а изменять стратегию управления этими устройствами очень просто - достаточно поместить в область векторов прерываний адрес программы, обслуживающей новое устройство.

Таким же точно образом устанавливаются в систему и резидентные программы.

Работать с векторами прерываний можно двумя способами. Первый способ - это прямая запись и чтение оперативной памяти в области векторов прерываний. Второй и более удобный способ - использование сервисного прерывания DOS INT 21H, функция 35H которого позволяет считать вектор прерывания в двухсловный элемент памяти,а функция 25H позволяет установить нужный вектор прерывания по адресу, помещенному в двухсловный элемент памяти.

Структура резидентной программы

В отличие от обычных программ, подавляющая часть кода которых выполняется в интервале с момента запуска до завершения, резидентные программы состоят из двух частей: собственно резидентной и вспомогательной части. При запуске программы выполняется только вспомогательная часть, подготавливающая резидентную часть к активизации.

Вспомогательная часть выполняет следующие действия:

1) сохраняет векторы "захватываемых" прерываний во внутренних переменных;

2) устанавливает эти векторы по адресу резидентной части программы;

3) инициализирует внутренние переменные резидентной части;

4) завершает работу программы без удаления резидентной части из памяти компьютера.

Реализация этих действий на низком уровне представляет собой:

1) вызов 35-й функции прерывания DOS INT 21H (GET VECTOR), которая позволяет сохранить вектор прерывания в двухсловной метке;

2) вызов 25-й функции прерывания DOS INT 21H (SET VECTOR),которая позволяет установить вектор прерывания по известному адресу;

3) присвоение начальных значений внутренним переменным в зависимости от их назначения в программе;

4) вызов прерывания DOS 27H (KEEP), которое позволяет завершить выполнение программы без удаления резидентной части из памяти компьютера.

С того момента как резидентная часть программы находится в ОЗУ, и на нее указывает вектор "захваченного" прерывания, она может быть активизирована путем вызова этого прерывания.

Построение резидентной части может осуществляться по трем схемам выполнения (рис. 1).

В схеме "А" сначала вызыва-

Рис. 1. Схемы выполнения резидентной части

ется системная обработка, а затем производится добавочная пользовательская обработка, написанная программистом. Эта схема подходит в том случае, когда программист хочет чем-либо дополнить системную обработку. Например, можно заставить машину менять по нажатию клавиш Ctrl-Alt атрибуты яркости символов на противоположные (пример 1).

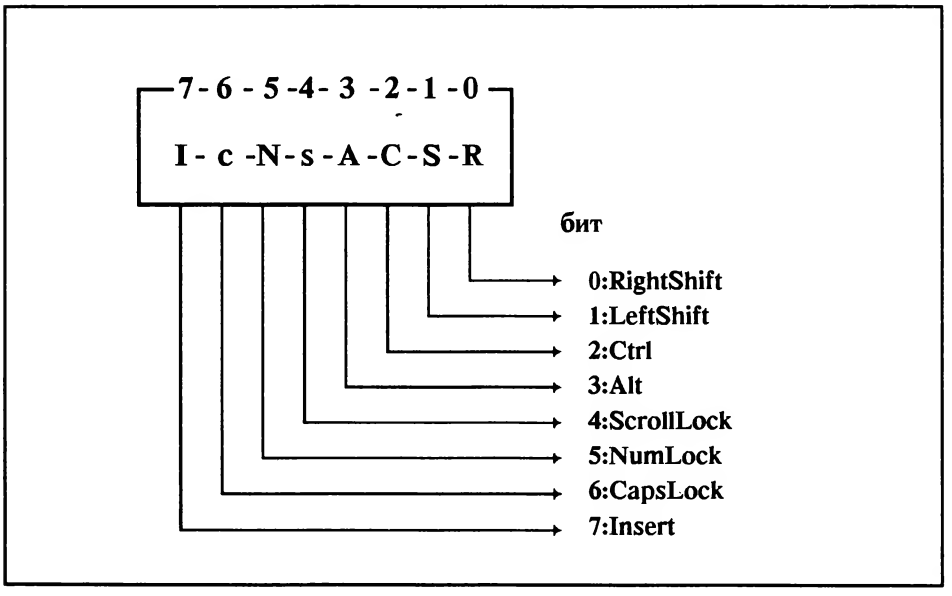


Рис. 2. Битовые флаги состояния клавиатуры

Программа, построенная по схеме "B", выполняет сначала пользовательскую обработку и только потом системную. Этот случай пригоден для изменения содержимого регистров или осуществления каких-либо действий перед системной обработкой. Схемой "B" пользуются все вирусные программы, отслеживающие прерывание запуска программы для того, чтобы присоединиться к ее концу. В примере 2 программа, написанная по схеме "B", при выводе по прерыванию на экран символа "пробел", заставляет систему выводить его на красном фоне. Данная программа перехватывает вектор сервисного прерывания INT 10H, функции 09H и 0AH которого служат для вывода символов на экран с текущей позиции курсора. Когда вызывается одна из этих функций, резидентная программа проверяет номер символа, выводимого на экран (он находится в регистре AL), и в зависимости от результата проверки переустанавливает цвет символа (регистр BL). После проверки производится переход на системную обработку, выводящую символ на экран с установленными при пользовательской обработке параметрами, и возвращается управление основному процессу.

Схема "C" полностью исключает системную обработку и заменяет ее пользовательской. При необходимости вообще исключить обработку данного прерывания резидентная часть должна состоять лишь из операторов засылки в порт 20H сигнала о конце прерывания (код 20H) и оператора возврата IRET. Пример 3 иллюстрирует рассматриваемую схему. Программа использует пользовательское прерывание по таймеру INT 1CH и выводит в левом верхнем углу значение счетчика. Системный таймер генерирует прерывание INT 1CH ("тикает") приблизительно 18,5 раз в секунду. Программа дожидается десяти таких "тиков" и увеличивает номер символа, выводимого в левом верхнем углу экрана, на единицу.

Активизация резидентных программ по горячим клавишам

Самым распространенным способом активизации резидентных программ по прерыванию является вызов по так называемым "горячим" клавишам ("hot keys"),т.е. по заранее установленной комбинации клавиш.

Легче всего обрабатывать сочетания клавиш Ctrl, Alt, LeftShift и RightShift. Резидентная программа, вызываемая по "горячим" клавишам, обычно устанавливается на прерывание INT 09H, хотя может устанавливаться и на прерывание INT 16 H. В начале резидентной части необходимо проверить байт памяти, находящийся по адресу 0000:0417H, в котором хранятся битовые флаги состояния клавиатуры, т.е. информация о нажатых в данный момент управляющих клавишах (рис.2). Если значение бита равно 1 (бит "взведен"), значит соответствующая клавиша в данный момент нажата или включен соответствующий режим.

Пример 1 иллюстрирует резидентную программу, использующую вызовы по "горячим" клавишам. Программа перехватывает аппаратное прерывание INT 09H и производит сначала системную обработку (см. схему "A" на рис. 1), затем проверяет флаги клавиатуры. Если флаги клавиатуры указывают на нажатие клавиш Ctrl и Alt,то программа осуществляет предписанные ей действия и передает управление основно-

му процессу.

Если необходимо использовать для активизации резидентной программы дополнительные (не входящие в байт флагов состояния клавиатуры) клавиши, то следует добавочно использовать порт клавиатуры. Фрагмент резидентной части, активизирующейся по сочетанию клавиш Ctrl-Alt-R, приведен на рис. 3.

Некоторым резидентным программам для нормальной работы необходимо захватывать более одного вектора прерываний. Примером таких программ являются русифика-

```
PROCESS:
        PUSHF                   ;НАЧАЛО РЕЗИДЕНТНОЙ ЧАСТИ
        CALL    CS:OLD_VEC      ;Имитация прерывания
        PUSH    AX              ;Вызов системной обработки
        PUSH    ES              ;Сохранение в стеке регистров
        IN      AX,60H          ;AX и ES
        CMP     AX,19           ;Чтение порта клавиатуры в AX
        JNZ     A1              ;Проверка нажатия клавиши "R"
        MOV     AX,0            ;если нет , то возврат управления
        MOV     ES,AX           ;Чтение флагов состояния
        MOV     AL,ES:[0417H]   ;клавиатуры в регистр AL и
        AND     AL,00001100B    ;Определение нажаты ли клавиши
        CMP     AL,00001100B    ;Ctrl-Alt, если нет , то
        JNZ     CMP             ;возврат управления
          .
          .
          .                     ;дальше как в примере 1
```

▲ Рис. 3.

Фрагмент резидентной части,активизирующейся при нажатии сочетания клавиш Ctrl-Alt-R

торы. Эти программы перехватывают прерывание INT 09H и с его помощью отслеживают момент смены режимов "Рус/Лат" по "горячим" клавишам, а также перехватывают функцию 00H прерывания INT 10 H для загрузки в знакогенератор шрифтов различных размеров при смене одного видеорежима на другой (к примеру, при смене текстового режима 25*80 со шрифтом 8x14 точек на текстовый режим 42*80 со шрифтом 8x8 точек).

Особенности программирования

Правильное построение резидентной и вспомогательной частей обеспечивает нормальную активизацию и работу фонового процесса, но после его завершения программу ожидает еще один важный этап - возврат управления основному процессу, во время выполнения которого была активизирована резидентная программа. Для успешного возврата в основной процесс необходимо восстановить значения всех регистров, которые были использованы во время работы резидентной части. Удобнее всего это сделать, поместив в начале работы фонового процесса значения этих регистров в стек, а в конце работы восстановить прежние значения, взяв их из стека.

Если регистры не будут восстановлены, то нормальный возврат в основной процесс не произойдет. Это скорее всего приведет к краху системы. Такая ошибка довольно часто допускается программистами, начинающими работать над написанием резидентных программ.

В резидентных программах необходимо аккуратно работать со стеком, так как при недостаточном его размере будет портиться память, находящаяся после резидентной программы.

К непредсказуемым последствиям приводит также несбалансированность операторов PUSH (положить в стек) и POP (извлечь из стека).

Все резидентные программы обработки прерываний должны заканчиваться командой возврата IRET,так как она кроме возврата осуществляет восстановление слова состояния процессора (PSW).

Может сложиться впечатление, что писать резидентные программы можно только на языках низкого уровня. Пример 4 опровергает это представление. Более того, транслятор языка TURBO PASCAL дает возможность реализации рассмотренных выше функций загрузки и чтения векторов прерываний на высоком уровне с помощью собственных библиотечных процедур. При использовании транслятора TURBO PASCAL программист не обязан следить за восстановлением регистров, так как это осуществляется автоматически при выходе из подпрограмм типа "Interrupt".

Написание резидентных программ ошибочно считается в среде пользователей и программистов "высшим пилотажем". Каждый программист, понимающий процессы, происходящие в компьютере, может достичь успехов в этой области.

; **Пример 1.** Резидентная программы, которая инвертирует яркость
; всех символов на экране при нажатии сочетания клавиш Ctrl-Alt.

```
CODE  SEGMENT                 ;Начало сегмента кода
      ORG     100H            ;Смещение 100h для программы типа COM
      ASSUME  CS:CODE,DS:CODE ;Назначение сегментов
BEGIN: JMP    START           ;Переход на выполнение вспомогательной части
      OLD_VEC LABEL   DWORD
      IP_OLD  DW      ?       ;Определение FAR метки
      CS_OLD  DW      ?
PRO:                          ;НАЧАЛО РЕЗИДЕНТНОЙ ЧАСТИ
      PUSHF                   ;Имитация прерывания
      CALL    CS:OLD_VEC      ;Вызов системной обработки
      PUSH    AX              ;Сохранение в стеке регистров AX и ES
      PUSH    ES
      MOV     AX,0            ;Чтение флагов состояния
      MOV     ES,AX           ;клавиатуры в регистр AL и
      MOV     AL, ES:[041 7H] ;определение нажаты ли клавиши
      AND     AL,00001100B    ;Ctrl-Alt, если нет , то
      CMP     AL,00001100B    ; возврат управления
      JNZ     A1
      PUSH    CX              ;Сохранение в стеке регистров
      PUSH    BX              ; CX , BX и SI
      PUSH    SI
      MOV     AX,0B800H       ;Адрес видеопамяти в ES
      MOV     ES,AX
      MOV     SI,1            ;Подготовка к инверсии яркости
      MOV     CX,2000
CYC:  MOV     AL,ES:[SI]      ;Чтение байта видеопамяти
      XOR     AL,00001000B    ;Инверсия яркости ( 3-й бит )
      MOV     ES:[SI],AL      ;Запись байта на место
      INC     SI              ;Увеличение смещения на 2
      INC     SI
      LOOP    CYC             ;Если не конец экрана , то продолжить инверсию
      POP     SI              ;Восстановление регистров,
      POP     BX              ;испорченных во время
      POP     CX              ;работы резидентной части
A1:   POP     ES
      POP     AX;
      IRET                    ;Возврат управления
START:                        ;НАЧАЛО ВСПОМОГАТЕЛЬНОЙ ЧАСТИ
      MOV     AX,3509H        ;Чтение вектора 09H
      INT     21H
      MOV     IP_OLD,BX       ;Запись вектора 09H в метку
      MOV     CS_OLD,ES
      MOV     AX,2509H        ;Установка вектора 09H на адрес
      LEA     DX,PRO          ;резидентной процедуры PRO
      INT     21H
      LEA     DX,START        ;Адрес конца резидентной части
      INT     27H             ;KEEP
CODE  ENDS                    ;Конец сегмента кода
      END     BEGIN           ;Конец программы
```


; **Пример 2.** Резидентная программа , которая заставляет
; систему выводить пробел на экран на красном фоне.

```
CODE  SEGMENT
      ORG     100H
      ASSUME  CS:CODE,DS:CODE
BEGIN: JMP    START
      OLD_VIDEO LABEL   DWORD
      IP_OLD  DW      ?
      CS_OLD  DW      ?
PRO:                          ;НАЧАЛО РЕЗИДЕНТНОЙ ЧАСТИ
      CMP     AH,09H          ;Проверка, 9-я ли это функция:
      JZ      A1              ;если да, то пользовательская обработка
      CMP     AH,0AH          ;Проверка, 10-я ли это функция:
      JZ      A1              ;если да, то пользовательская обработка;
      JMP     CS:OLD_VIDEO    ;если нет, то системная обработка
A1:
      CMP     AL,32           ;Сравнить выводимый символ с пробелом:
      JZ      A2
      MOV     BL,0FH          ;если пробел, то красный фон;
      JMP     A3
A2:   MOV     BL,4FH          ;если не пробел, то черный фон
A3:   MOV     AH,09H          ;Функция 09H
      JMP     CS:OLD_VIDEO    ;Системная обработка

START:                        ;НАЧАЛО ВСПОМОГАТЕЛЬНОЙ ЧАСТИ
      MOV     AX,3510H        ;Чтение вектора 10H
      INT     21H
      MOV     IP_OLD,BX       ;Запись вектора 10H в метку
      MOV     CS_OLD,ES
      MOV     AX,2510H        ;Установка вектора 10H на адрес
      LEA     DX,PRO          ;резидентной процедуры PRO
      INT     21H
      LEA     DX,START        ;Адрес конца резидентной части
      INT     27H             ;KEEP
CODE  ENDS                    ;Конец сегмента кода
      END     BEGIN           ;Конец программы
```

```
; Пример 3. Резидентная программа, выводящая в левом углу экрана
; символ , номер которого увеличивается через каждые 10 тиков таймера на 1

-CODE  SEGMENT
       ORG     100H
       ASSUME  CS:CODE,DS:CODE
BEGIN: JMP     START
       INT_TIM LABEL  DWORD
       IP_TIM  DW     ?
       CS_TIM  DW     ?
       TIK     DB     ?
       NCH     DB     0
PROCESS:                        ;НАЧАЛО РЕЗИДЕНТНОЙ ЧАСТИ
       CLI                      ;Запрещение прерываний
       INC     TIK              ;Увеличение счетчика тиков
       CMP     TIK,10           ;Сравнение его с 10:
       JL      A1               ;если меньше, то конец прерывания
       PUSH    ES               ;Сохранение регистров в стеке
       PUSH    AX
       MOV     AX,0B800H        ;Вывод символа по номеру NCH
       MOV     ES,AX            ;прямой записью в видеопамять
       XOR     AL,AL
       MOV     TIK,AL
       INC     NCH
       MOV     AL,NCH
       MOV     ES:[0],AL
       MOV     AL,1EH
       MOV     ES:[1],AL
       POP     AX               ;Восстановление регистров
       POP     ES
A1:    STI                      ;Разрешение прерываний
       PUSH    AX
       MOV     AX,20H           ;Сигнал о конце прерывания
       OUT     AX,20H
       POP     AX
       IRET                     ;Возврат управления

START:                          ;НАЧАЛО ВСПОМОГАТЕЛЬНОЙ ЧАСТИ
       MOV     AX,351CH         ;Сохранение вектора 1CH
       INT     21H
       MOV     IP_TIM,BX
       MOV     CS_TIM,ES
       MOV     AX,251CH         ;Установка вектора 1CH
       LEA     DX,PROCESS       ;по адресу метки PROCESS
       INT     21H
       MOV     AL,0             ;Установка начального
       MOV     TIK,AL           ;значения переменной TIK
       LEA     DX,START         ;Установка адреса конца
       INT     27H              ;резидентной части и KEEP
CODE   ENDS
       END     BEGIN
```

```
(* Пример 4. Резидентная программа, которая при
нажатии клавиш Ctrl-Alt генерирует звук *)

{$M 3000,0,3000}                    (* Резервирование памяти *)

Uses Crt,Dos;

Program Resident_Example ( Input , Output );
var
  I:word;
  Ch:char;
  Int9:procedure;
  logr:boolean;

{$F+}

(* РЕЗИДЕНТНАЯ ЧАСТЬ ПРОГРАММЫ *)

Procedure PressButton(Flags,CS,IP,AX,BX,CX,DX,SI,DI,ES,DS,BP:word);interrupt;
var
  B:byte;
  I:word;
begin
  Inline($9c);                      (* Имитация прерывания ( PUSHF ) *)
  Int9;                             (* Вызов системной обработки *)
  B:=Mem[$0000:$0417];              (* Чтение флагов клавиатуры *)
  If ((B and 12) = 12) and (Logr) then  (* Проверка нажатия клавиш Ctrl-Alt *)
  begin
    Logr:=False;
    I:=500;
    While I<=1000 do                (* Программирование динамика *)
    begin
      Sound(I);
      Delay(1);
      Inc(I,20);
    end;
    Nosound;
    Logr:=True;
  end;
  if (b and 5)=5 then               (* Проверка нажатия RightShift-Ctrl *)
    SetIntVec($9,Addr(Int9));       (* Восстановление захваченного вектора *)
end;
                                    (* Возврат управления *)
{$F-}
(* ВСПОМОГАТЕЛЬНАЯ ЧАСТЬ ПРОГРАММЫ *)
begin
  GetIntVec($9,@Int9);              (* Сохранение вектора 09H *)
  SetIntVec($9,Addr(PressButton));  (* Установка вектора 09H *)
  Logr:=True;                       (* Инициализация внутренней переменной *)
  Keep(0);                          (* Выход с сохранением в памяти *)
end.
```

Несмотря на широкое распространение IBM PC, для рядового пользователя остается много темных мест в архитектуре компьютера и процессах, происходящих в машине.

Управление загрузкой ОС на IBM PC

В. Аношин

Несмотря на широкое распространение ПЭВМ типа IBM PC и обилие популярной литературы по ним, для рядового пользователя остается много темных мест в архитектуре и процессах, происходящих в машине. Одним из таких темных мест является процесс загрузки машины, т.е. то, что происходит с момента включения питания и до появления приглашения операционной системы (ОС).

В момент включения питания или получения сигнала reset процессор производит самотестирование и передает управление программе инициализации BIOS. После тестирования оборудования и памяти BIOS проверяет наличие дисков, с которых может быть произведена загрузка ОС. При отсутствии дискеты в дисководе A: и жесткого диска, BIOS передает управление ROM BASIC. Поиск загрузочных дисков осуществляется в следующем порядке : сначала проверяется наличие дискеты, а затем винчестера. С первого же обнаруженного диска BIOS производит попытку чтения первого сектора нулевой дорожки нулевой головки (стороны). При удачном чтении этого сектора BIOS записывает его по адресу 0000:7C00h и осуществляет межсегментный переход по нему. На этом заканчивает работу программа инициализации.

BIOS не делает различия между дискетой и винчестером. Принципиальной разницы действительно нет , но есть разница в объеме хранимой информации. Винчестер, в отличие от дискеты - это "пакет" жестких, непрерывно вращающихся дисков. То, что эти диски не подлежат замене, позволило заключить их в герметичный корпус и обеспечить гораздо более точное позиционирование магнитных головок. Это резко повысило плотность записи. Число дорожек возросло с десятков у дискет до сотен и даже тысяч у твердых дисков, число секторов на одной дорожке - до 17, 26 или 35 , а за счет увеличения числа самих дисков, увеличилось число "сторон" (т.е. число магнитных головок). Как следствие, уве-

Смещение	Длина	Содержимое
000H	1BEH	Код загрузки и выполнения корневого сектора активного раздела (загрузчик ОС)
1BEH	10H	Элемент раздела 1
1CEH	10H	Элемент раздела 2
1DEH	10H	Элемент раздела 3
1EEH	10H	Элемент раздела 4
1FEH	02H	0AA55H - подпись таблицы разделов

Таблица 1. Формат загрузочного сектора винчестера

личилось и общее число секторов. Если на стандартной дискете (360 Кбайт) число секторов равно (число сторон)*(число дорожек)*(число секторов на дорожке)=2*40*9=720, то на винчестере (43 Мбайт) это число равно 8x640x17=87040.

При использовании функций BIOS, работающих с дисками, им, кроме других параметров, необходимо передавать и местоположение нужного сектора, задав номер диска, головки, дорожки и номер сектора на дорожке в виде значений 16-разрядных регистров PC. Максимальное число, которое можно записать в такой регистр, равно 65535, поэтому обращение к любому сектору на винчестере не вызовет никаких проблем. Это обращение называется обращением к физическим секторам, так как мы явно указываем на местонахождение сектора. Но большинство операционных систем обычно использует другой метод идентификации сектора. Например, функции DOS работают с логическими секторами и дисками. Логический номер диска - это порядковый номер его буквенного обозначения в латинском алфавите (A: - 1; B: - 2...), а логический номер сектора - это номер сектора относительно начала логического диска. Такой подход позволяет программисту, использующему функции DOS, не заботиться о физической организации диска и дает возможность применить такое средство операционной системы, как виртуальные диски.

Логические номера диска и сектора также передаются функциями DOS как значения 16-разрядных регистров. Учитывая, что стандартный размер сектора в DOS 512 байт, нетрудно подсчитать, что максимальный размер логического диска, целиком доступного через функции DOS, соответствует 65535*512=33553920 байт или 32 Мбайт. К тому же не всегда бывает удобно иметь все логические диски такого размера. Поэтому принято разбивать все дисковое пространство винчестера на несколько логических дисков (разделов). Каждый из разделов принадлежит какой-либо операционной системе, и его внутренняя структура полностью определяется этой операционной системой. Общим является только загрузчик ОС.

Информация о разделах хранится в таблице разделов винчестера (Partition Table), находящейся в первом секторе нулевой стороны нулевой дорожки. Эта таблица содержит данные о местонахождении разделов и об их размере. Сектор, содержащий эту таблицу, называется стартовым (загрузочным) сектором винчестера. Кроме таблицы разделов в нем находится загрузочная программа. Стандартный формат стартового сектора приведен в табл. 1 и 2.

Каждый элемент таблицы разделов описывает один раздел. Раздел начинается с сектора, указанного в таблице, продолжается до конца текущей дорожки, затем переходит на сторону с большим номером, затем на 1-й сектор 0-й стороны следующей дорожки и т.д. до сектора, описанного как конец раздела. Этот порядок следования секторов раздела соблюдается при логической нумерации секторов.

В случае с дискетой в загрузочном секторе находится информация о физическом размере диска, количестве секторов, плотности формата, количестве дорожек и т.п. Кроме того, в этом же секторе находится программа загрузки ОС, использующая эту информацию. Программа проверяет наличие операционной системы на диске и производит ее загрузку. Формат загрузочного сектора дискеты полностью идентичен формату загрузочного сектора раздела винчестера. В случае с винчестером BIOS при перезагрузке машины считывает не загрузочный сектор раздела, а загрузочный сектор винчестера, который и управляет дальнейшей загрузкой. Его цель - найти по таблице разделов активный раздел и передать ему управление, как это делает BIOS с дискетой.

Проследим за работой простой загрузочной

Смещение	Длина	Содежимое
00	01	Флаг загрузки : 00H - не активен, 80H - активен
01	01	Начало раздела : номер головки
02	02	Начало раздела : сектор/дорожка корневого сектора
04	01	Код системы : 0 - неизвестная, 1 - DOS, 12-ти битный FAT, 2 - DOS, 16-ти битный FAT
05	01	Конец раздела : номер головки
06	02	Конец раздела : сектор/дорожка последнего сектора
08	04	Относительный номер начального сектора
0CH	04	Размер (число секторов)

Таблица 2. Структура элемента раздела в таблице разделов

программы (см. приложение).

1. Так как загрузка этой программы производится не операционной системой, она сама должна заботиться о своих сегментах, т.е. об информации в регистрах DS, ES, SS, SP. Ввиду того что мы используем простейшую модель памяти,где сегментом данных является кодовый сегмент, во все регистры сегментов мы записываем значение регистра CS - 0.

2. Так как основная функция загрузочного сектора винчестера заключается в том, чтобы на основе анализа таблицы разделов найти раздел, содержащий ОС и помеченный, как активный, следует загрузить из него загрузочный сектор раздела, имитируя работу программы инициализации BIOS. Он должен загрузить этот сектор тоже по адресу 0000:7C00h. Для этого нужно переместить свой собственный код на другой адрес и передать ему управление.

3. В завершение загрузки необходимо считать код из загрузочного сектора активного раздела и передать на него управление.

Размер кода, выполняющего эти действия, очень невелик - всего несколько десятков байт, в то время как в распоряжении программиста находится весь загрузочный сектор (за исключением места, отведенного для таблицы разделов), т.е. 512-50=462 байта! Этого места достаточно для того, чтобы заставить загрузочную программу выполнять кроме обычных и несвойственные ей функции:

- запрос пароля (функцию защиты);
- запрос загрузочного раздела (в зависимости от выбора пользователем операционной системы загрузка с различных разделов винчестера, а не с раздела, помеченного как активный);
- загрузку системы с других устройств (например, со второго винчестера), если аппаратная организация машины не позволяет делать это автоматически;
- проверку корректности информации в таблице разделов;
- защиту от вирусов типа "Stoned" (вирусы, "живущие" в загрузочном секторе).

Проследим за работой простой загрузочной программы

Предлагаемая простейшая программа на языке Ассемблера иллюстрирует содержание этой статьи - она позволяет записать в загрузочный сектор винчестера свой собственный код загрузчика, не портя содержимого таблицы разделов. Загрузчик, кроме обычных функций, выполняет еще функцию запроса пароля.

Программа может быть набрана как текстовый файл с именем PART.ASM. Для получения исполнимого файла необходим компилятор Ассемблера, программы link.exe и exe2bin.exe.

Для компиляции Ассемблером фирмы Microsoft необходимо набрать следующие команды:

```
>masm part.asm

>link part

>exe2bin part.exe part.com
```

Для запуска полученного исполнимого модуля:

```
>part пароль
```

где "пароль" - пароль, который вы хотите установить.

Для того чтобы снять установленный ранее пароль, надо запустить программу без параметров:

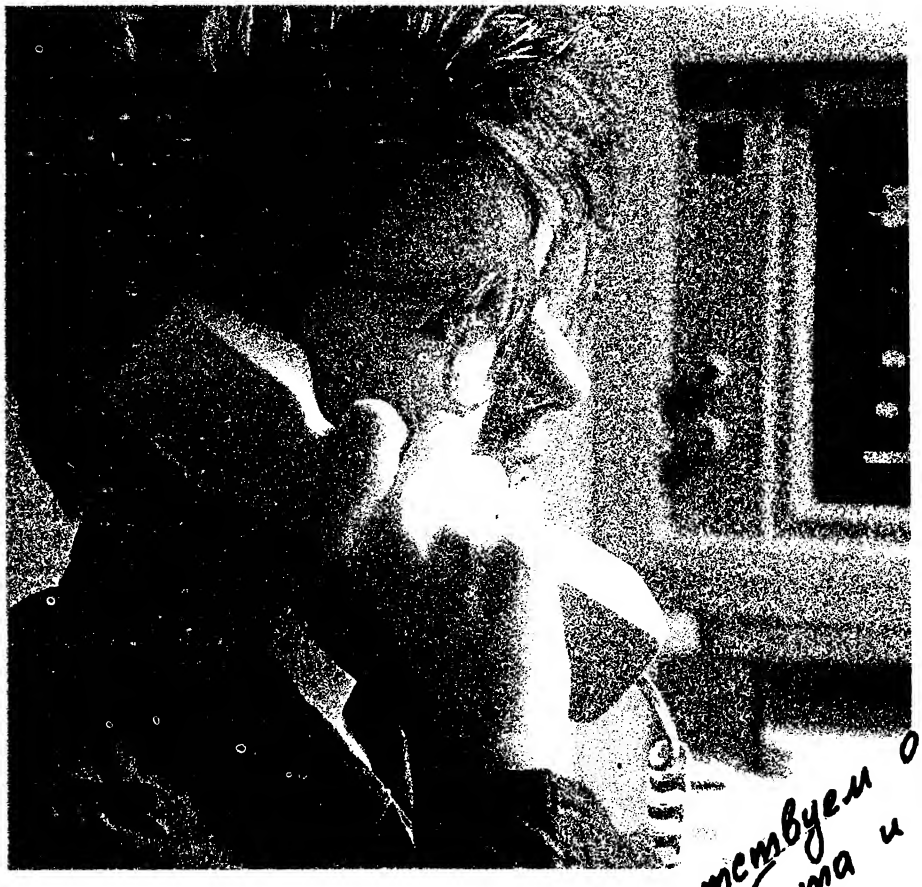
```
>part пароль
```

Примечание: Так как программа загрузчика будет загружаться в память до операционной системы, драйверов экрана и клавиатуры, вывод сообщений и ввод пароля могут быть осуществлены ТОЛЬКО буквами и символами, определенными для стандартного монитора. Только при наличии русифицированного ПЗУ у вашего монитора возможен вывод сообщений по-русски.

Приложение. Простая загрузочная программа.

```
NEW_SEG EQU     0040H
OLD_SEG EQU     07A0H
PART_SEG SEGMENT PARA   PUBLIC
        ASSUME  CS:PART_SEG,DS:PART_SEG
        ORG     100H            ;Стартовый адрес для "COM" файла
;*********************************************************
;****** Программа установки загрузчика на винчестер *****
;*********************************************************
MAIN    PROC    NEAR
        PUSH    DS              ;Сохраним регистр данных
        MOV     AX,CS           ;Запишем значение cs
        MOV     DS,AX           ;в ds и es через ax
        MOV     ES,AX
;****** Узнаем, какой устанавливать пароль
        MOV     SI,5DH          ;Адрес пароля в командной строке
        LEA     DI,PASS         ;(до 8 символов)
        XOR     CX,CX
READ_P:
        LODSB
        CMP     AL,20H
        JE      M00
        CMP     AL,40H
        JNA     M0
        CMP     AL,5AH
        JA      M0              ;Если заглавная буква, то
        ADD     AL,20H          ;переведем ее в нижний регистр
M0:
        INC     CX
        STOSB
        JMP     READ_P
;****** Прочитаем с диска таблицу разделов
M00:
        MOV     LEN_PASS,CX
        MOV     DX,80H          ;Сторона 0 диск 80H
        MOV     CX,0001H        ;Дорожка 0 сектор 1
        LEA     BX,PART_OLD     ;Адрес буфера ввода
        CALL    GET_SECT        ;Подпрограмма чтения сектора
        LEA     SI,PART_T_OLD   ;Адрес старой таблицы разделов
        LEA     DI,PART_T       ;Адрес новой таблицы разделов
        MOV     CX,42H          ;Длина таблицы разделов
        REP     MOVSB           ;Скопировать таблицу разделов
;****** Запишем сектор
        MOV     DI,3            ;При ошибке повторить 3 раза
M01:    MOV     DX,80H          ;Сторона 0 диск 80H
        MOV     CX,0001H        ;Дорожка 0 сектор 1
        MOV     AX,0301H        ;Функция записи секторов
        LEA     BX,PART         ;Адрес загрузочного сектора винчестера
        INT     13H             ;Вызов прерывания BIOS
        JNC     M02             ;Переход, если нет ошибок записи
        XOR     AX,AX           ;Функция сброса дисковой системы
        INT     13H
        DEC     DI
        JNZ     M01
M02:
        POP     DS
        MOV     AX,4C00H        ;Функция выхода из программы
        INT     21H             ;Прерывание DOS
MAIN    ENDP
; *********************************************************
; ******      Собственно загрузчик винчестера        ******
; *********************************************************
        ORG     200H
PART    PROC    NEAR
        DB      0EAH
        DW      OFFSET M1       ;
        DW      OLD_SEG         ;JMP FAR OLD_SEG:M1

D1:     DB      '(C) 1991 by Anoshin Slava.',0DH,0AH
        DB      'Enter password:',0
D3:     DB      0DH,0AH,'Can''t load boot sector.',0
M1:     MOV     AX,CS
        MOV     DS,AX           ;Установим регистры сегментов
        MOV     ES,AX
        XOR     AX,AX           ;AX=0
        MOV     SS,AX
        MOV     SP,7C00H        ;Адрес начала стека
        LEA     SI,D1           ;Адрес запроса на пароль
        CALL    PUTSTR          ;Выведем запрос на пароль
GETPASS:
        LEA     DI,PART_OLD     ;Адрес буфера ввода вывода
GETSTR:
        XOR     AH,AH           ;AH=0 функция чтения символа
        INT     16H             ;с клавиатуры без отображения
        STOSB                   ;его на экране
        CMP     AL,0DH          ;Нажат <ВВОД> ?
        JNE     GETSTR          ;Повторять пока не нажат <ВВОД>
        LEA     DI,PART_OLD     ;Введенный пароль
        LEA     SI,PASS         ;Истинный пароль
```

```
        MOV     CX,LEN_PASS
        REPE    CMPSB           ;Сравним пароль,
        JNE     GETPASS         ;если не тот - вводим снова
        MOV     AX,NEW_SEG
        MOV     ES,AX           ;Сегмент назначения
        LEA     SI,PART
        MOV     DI,SI           ;Адрес, куда перемещаем загр. сегмент
        MOV     CX,0200H        ;Длина загрузочного сегмента
        CLD
        REP     MOVSB           ;Копируем сегмент
        DB      0EAH            ;Передаем управление на
        DW      OFFSET M2       ;скопированный код
        DW      NEW_SEG         ;(JMP FAR NEW_SEG:M2)
;****** Адрес последнего элемента таблицы разделов
M2:     MOV     SI,((OFFSET PART_T)+30H)
M3:
        CMP     BYTE PTR [SI],80H;Ищем активный раздел
        JE      M4
        SUB     SI,10H
        CMP     SI,OFFSET PART_T
        JAE     M3
        INT     18H             ;Если нет активного раздела, передадим
M4:                             ;управление ROM BASIC
;****** Считаем с диска загрузчик активного раздела по адресу 0000:7c00
        MOV     AX,0
        MOV     ES,AX           ;Сегмент назначения - 0
        MOV     DH,[SI+1]       ;Номер головки(стороны)
        MOV     CX,[SI+2]       ;Номера дорожки и сектора
        MOV     BX,7C00H        ;Адрес загрузки
        MOV     DL,80H          ;Номер диска
        CALL    GET_SECT
        DB      0EAH            ;Передадим управление
        DW      7C00H,0000H     ;загрузчику системы
            ;(JMP FAR 0000:7C00)
PART    ENDP


;*********************************************************
;******     Подпрограмма вывода строки на экран     ******
;******          Адрес строки - в регистре SI       ******
;*********************************************************
PUTSTR  PROC    NEAR
        LODSB                   ;Загрузить в AL выводимый символ
        CMP     AL,00H          ;0 - конец строки
        JE      M8
        MOV     BX,7
        MOV     AH,0EH          ;Функция вывода символа на экран
        INT     10H
        JMP     SHORT PUTSTR
M8:
        RET
PUTSTR  ENDP

LEN_PASS DW     0000H           ;Длина пароля
PASS:   DB      20      DUP(0)  ;Пароль
;*********************************************************
;******     Подпрограмма чтения сектора с диска     ******
;*********************************************************
GET_SECT PROC   NEAR
        MOV     DI,3            ;При ошибке повторить 3 раза
M10:
        MOV     AX,201H         ;Функция чтения 1 сектора с диска
        INT     13H
        JNC     M12             ;Выйти, если нет ошибки чтения
        XOR     AX,AX
        INT     13H
        DEC     DI
        JNZ     M10
M11:
        LEA     SI,D3           ;Вывод сообщения об
        CALL    SHORT PUTSTR    ;ошибке чтения с диска
M:
        JMP     SHORT M         ;Зацикливаем программу
M12:
        RET
GET_SECT ENDP
;*********************************************************
;******                Область данных             ******
;*********************************************************
        ORG     3BEH            ;Начало таблицы разделов
PART_T  LABEL   WORD
        ORG     400H            ;Буфер ввода-вывода
PART_OLD LABEL  WORD
        ORG     5BEH            ;Начало табл. разд. в вводимом секторе
PART_T_OLD LABEL WORD
PART_SEG ENDS
        END     MAIN
```



Приветствуем отделы сбыта и снабжения!

Если у Вас есть проблемы со сбытом или снабжением, сегодня же позвоните в

"ГРАНИТ"

Малое предприятие "ГРАНИТ" - это

- брокерская контора No.869 на Российской товарно - сырьевой бирже;

- представительство на Российской агропромышленной бирже,Владивостокской бирже "Азия", Уральской бирже, Казахстанской республиканской бирже.

- связи с десятками предприятий и организаций во всех регионах страны.

ПОМНИТЕ: "ГРАНИТ" ЖДЕТ ВАШЕГО ЗВОНКА!

по тел.(095) 236-03-69, 237-11-45.

История мировой компьютеризации. За 40 лет от раба до рабовладельца

50-ые годы

1951

Национальный комитет по переписи (США) получает первую ЭВМ "UNIVAC I" от фирмы "Remington Rand"

Фирма "IBM" решает приступить к производству ЭВМ "701", первого электронного двоичного компьютера

1952

Компьютер "UNIVAC" успешно предсказывает итоги президентских выборов 1952 года

1953

Первая модель магнитной оперативной памяти разработана в Массачусетском технологическом институте (MIT)

1954

Фирма "General Electric Co." впервые успешно использовала компьютер "UNIVAC" I для операций с платежными ведомостями

Фирма "IBM" объявила о выпуске ЭВМ "650" и приступила к разработке машины "704"

1955

"IBM" начала создавать собственный экспериментальный компьютер "STRETCH"

Компания "General Electric Co." начинает заниматься компьютерным бизнесом

1957

Фирма "Honeywell" объявляет о выпуске гигантского компьютера "Datamatic 1000"

Учреждены компании "Control Data Corp." и "Digital Equipment Corp."

Первый компилятор для языка Fortran стал использоваться на "IBM-704"

1958

Сотрудник "Texas Instruments" Колби изобрел первую интегральную микросхему на основе германия

Система противовоздушной обороны "SAGE" установлена на базе военно-воздушных сил McGuire, Нью-Джерси, США

Добровольная организация "CODASYL" приступила к разработке языка высокого уровня COBOL

60-ые годы

1961

Роберт Нойс запатентовал интегральную микросхему на основе кремния

1963

Код ASCII становится стандартом

Фирма "General Electric Co." реализовала объединенную систему размещения данных, первую коммерческую систему управления базами данных

Первая система автоматизированного проектирования (CAD) начинает использоваться на конвейерах фирмы "General Motors"

1964

"IBM" приступает к разработке нового семейства машин типа "360"

Система разделения времени и язык Бейсик разрабатываются в колледже города Дартмута

1965

Фирма "Digital Equipment Corp." объявила о создании "PDP-8", одного из первых мини-компьютеров

Принимается поправка Брукса - закон, организующий рынок компьютерных изделий

1968

Фирма "Intel Corp." заявила о выпуске собственного микропроцессора

Первый патент на программное обеспечение (процесс сортировки) получил Мартин Гетц

Эдсгер Дейкстра разработал концепцию структурного программирования

1969

Начинают эксплуатироваться сети типа APRANET, в которых использовалась передача информации в виде пакетов

Сотрудники "Bell Lab's" Кен Томпсон и Деннис Ричи придумывают операционную систему UNIX

Появляется суперкомпьютер "CDC 7600" фирмы "Control Data Corp"

70-ые годы

1971

Микропроцессор, придуманный Тедом Ноффом, поступает на рынок под маркой "Intel 4004"

1972

Появляются суперкомпьютер Роберта Крея и одноименная фирма

1973

Фирмы "Philips", "Siemens", "CII" учреждают консорциум "Unidata"

Сотрудник фирмы "Xerox" Роберт Меткалф разработал сеть Ethernet

1974

Сотрудник фирмы "IBM" Джон Кук разработал первую RISC-машину

1975

Фирма "Philips" перестала производить большие ЭВМ

Фирма "Xerox" отказалась от выпуска больших ЭВМ

Корпорация "Wang Corp." выпустила программные продукты по обработке текстовой информации.

Фирма "Digital Research" представила операционную систему CP/M

1976

Были организованы фирмы "Microsoft" и "Apple Computer"

Поступил в продажу компьютер "Apple I"

1977

С появлением компьютеров "Altair 8800", "TRS-80 PC" и "PET PC" (Commodore) возник рынок персональных компьютеров

1979

На рынке появилась VisiCalc (электронная таблица), созданная Робертом Бриклином

Объемы IBM перевалили за 20 биллионов долларов; "Digital Research" - за 2 биллиона

80-ые годы

1981

Был создан Compaq - один из первых "lap-top"

"IBM" представила собственный персональный компьютер с операционной системой MS-DOS

1982

Фирма "Sun" начала выпускать рабочие станции.

При попытке получить документы фирмы "IBM" арестованы агенты компаний "Hitachi" и "Mitsubishi"

"Lotus Development Corp." выпустила электронную таблицу Lotus 1-2-3

1984

Фирма "Apple" выпустила персональный компьютер "Macintosh PC"

Компания "Storage Technology" намеревается продать свое производство больших ЭВМ

Фирма "AT&T" заявила о твердом выполнении стандарта UNIX

1986

"NEC", объединившись с компаниями "Groupe Bull" и "Honeywell", становится третьей крупнейшей компанией мира в компьютерном бизнесе

1987

"IBM" представила машины семейства PS/2

"AT&T" дебютировала со своим персональным компьютером, разработанным на фирме "Olivetti"

1988

Фирма "Fujitsu" выплатила 237 миллионов долларов за несанкционированное использование программного обеспечения фирмы IBM

90-ые годы

"Hitachi" стала производить две IBM-совместимых ЭВМ

"IBM" представила машины семейства "390"

По материалам журнала "Datamation"

Мы открываем в нашем журнале рубрику "Этюды для программистов". Публикуемые в этой рубрике материалы предназначены как для любителей поломать голову над занимательными задачами, так и для программистов, стремящихся осознать математические основы своего ремесла, систематизировать и обосновать используемые ими приемы построения и анализа программ и открыть для себя новые.Материал каждого публикуемого этюда будет расположен следующим образом.

1. Постановка задачи. Мы будем стремиться к тому, чтобы за внешней простотой и понятностью формулировки задачи скрывалась содержательная проблема, решение которой вело бы к осознанию некоторых основополагающих принципов программирования.

2. Текст программы. Мы будем сосредотачиваться на существе алгоритма и поэтому уделять меньше внимания таким, несомненно важным при реальном программировании вопросам, как контроль правильности входной информации и красивая выдача результатов.

3. Анализ решения. Возможное содержание этого раздела: доказательство правильности программы, вывод ее свойств, четкое выявление лежащей в основе задачи проблемы и применяемых приемов решения, проведение аналогий с другими задачами (алгоритмами), демонстрация различных вариантов решения и обьяснение сделанного выбора.

4. Задачи для самостоятельного решения в развитие главной темы этюда.

После того как вы прочитаете формулировку разбираемой в этюде задачи, мы советуем прервать чтение и попытаться найти решение самостоятельно, ибо только в этом случае можно оценить трудность задачи и изящество решения (или отсутствие таковых), а также получить удовлетворение в случае, если ваше решение окажется лучше. Мы надеемся на то, что ваши письма помогут нам в реализации поставленных целей и послужат источником новых тем и задач. Наиболее интересные из писем будут находить отражение на наших страницах.

Ведущий рубрики - Д.Бройтман

Этюд 1. Беспорядок в строю или анатомия одного цикла

Д.Бройтман

Новобранцы выстроены в шеренгу. По команде "напра-во" одни из них поворачиваются направо, другие - налево. Далее каждый действует следующим образом: увидев перед собой лицо соседа, он ровно через секунду поворачивается кругом.

Примечание. Формулировка задачи заимствована из книги "Простое и сложное в программировании". М.: Наука, 1988.

Задание.

1. Напишите программу, моделирующую поведение новобранцев в строю.

2. Докажите, что при любой начальной позиции новобранцев (под начальной мы будем понимать позицию, возникшую после первого поворота) повороты в конце концов прекратятся.

3. Опишите все возможные заключительные позиции шеренги новобранцев. Докажите, что других быть не может.

Ниже приведен текст программы на языке Си, моделирующей поведение новобранцев (см. пример) .

Содержательная часть нашей программы состоит из цикла do ... while(turns()), реализующего предложенные правила поведения солдат.

Тело цикла: все пары солдат, стоящие лицом друг к другу, разворачиваются на 180 градусов.

Условие выхода из цикла: в шеренге нет ни одной пары солдат-соседей, стоящих лицом друг к другу.

Для доказательства завершимости программы нам надо показать, что при любой начальной позиции солдат повторение тела цикла в конце концов приведет к выполнению условия выхода.

Мы могли бы использовать для доказательства метод математической индукции. Для шеренги из одного солдата справедливость утверждения очевидна. Предположим, что оно справедливо для шеренги из n солдат, и покажем, что тогда оно справедливо и для шеренги из (n+1) солдат... Однако существует гораздо более изящное доказательство. Переформулируем нашу задачу следующим образом. Каждому солдату, повернувшемуся налево, поставим в соответствие цифру 0, повернувшемуся направо - цифру 1. Тогда шеренге из n солдат соответствует n-значное двоичное число. Паре солдат, стоящих лицом друг к другу, - пара цифр 10. Тело цикла - замена

Иногда полезно переформулировать исходную задачу в других терминах

Пример программы

```
#include <stdio.h>
#include <dos.h>          /* sleep( ) */
#include <string.h>       /* strlen( ) */

#define LEFT_TURN '<'     /* символ левого поворота */
#define RIGHT_TURN '>'    /* символ правого поворота */
#define MAX_LEN 60        /* максимальная длина шеренги */

#define INVERSION (row[i]==RIGHT_TURN)&&(row[i+1]==LEFT_TURN)

char row[MAX_LEN];        /* шеренга */
int len;                  /* длина шеренги */

/*-------------------------------*/
/* Ввод начальной позиции */
/*-------------------------------*/
void enter_init_pos(void)
{
  printf("Введите начальную позицию солдат в шеренге.\n");
  printf("Используйте символы %c (левый поворот) ",LEFT_TURN);
  printf("и %c (правый поворот).\n", RIGHT_TURN);
  printf("Длина шеренги <=%d.\n",MAX_LEN);
  printf("Начальная позиция: ");
  scanf("%s", row);
  len = strlen(row);
}
```

```
/*-----------------------------------------------------------------------*/
/* Функция разворачивает солдат, стоящих лицом друг */
/* к другу, кругом. Возвращает 1, если повороты были; */
/* в противном случае возвращает 0.                    */
/*-----------------------------------------------------------------------*/
char turns(void)
{
  int i;
  char b = 0;
  for(i = 0 ; i < len - 1 ; i++)
    if(INVERSION) {
      row[i] = LEFT_TURN;
      row[++i] = RIGHT_TURN;
      b = 1; }
  return(b);
}
/*-------------------------------------*/
/* Печать текущей позиции   */
/*-------------------------------------*/
void print_cur_pos(void)
{
  sleep(1);                /* пауза в 1 секунду */
  printf("%s\n", row);
}
main( )
{
  enter_init_pos( );
  do print_cur_pos( );
  while(turns( ));
  printf("Повороты завершены.\n");
  return(0);
}
```

всех пар 10 в числе на 01. Условие выхода из цикла - в числе отсутствует пара цифр 10.

Во-первых, заметим, что при замене 10 на 01 число единиц (также, как и число нулей) остается неизменным. Возвращаясь к нашей исходной формулировке, мы можем заключить, что число солдат, повернувшихся направо (налево), после выполнения тела цикла не меняется и остается равным числу солдат, которые повернулись направо (налево) в начальной позиции. Такое утверждение, справедливое перед каждым входом в цикл, называют *инвариантом цикла.*

Во-вторых, заметим, что замена пары 10 на пару 01

уменьшает величину двоичного числа не менее, чем на единицу.

В третьих, наше число всегда остается неотрицательным (точнее, оно всегда больше или равно 2**m - 1, где m - число единиц).

Итак, мы установили, что при каждом выполнении тела-цикла значение некоторого двоичного числа уменьшается как минимум на единицу, но так как это число не может стать отрицательным, цикл обязательно должен завершиться.

Наше доказательство иллюстрирует *общий метод*, часто используемый для доказательства завершимости работы программ.

Общий метод доказательства завершимости работы программ

Пусть x - числовая функция от переменных программы, вычисляемая при каждом входе в цикл, и справедливы следующие утверждения.

1. Существует числовая константа M, такая, что всегда x >= M.

2. При каждом выполнении тела цикла x уменьшается не менее, чем на единицу.

Тогда работа цикла обязательно завершится.

Если мы будем запускать нашу программу для разных начальных позиций шеренги из 6 солдат, то увидим, что встречаются лишь 7 заключительных позиций (<- солдат, повернувшийся влево, >- вправо):

1)<<<<<<
2)<<<<<>
3)<<<<>>
4)<<<>>>
5)<<>>>>
6)<>>>>>
7)>>>>>>

Докажем следующее утверждение о свойствах заключительных позиций: если в шеренге из n солдат в начальной позиции k <= n солдат повернулись налево, а (n - k) солдат - направо, то, когда повороты завершатся, левые k солдат шеренги окажутся повернувшимися налево, а правые (n - k) - направо.

<< .... < > ..... >>
---- k --- -- n - k --

Мы показали ранее, что при любой начальной позиции повороты в шеренге завершатся, то есть произойдет выход из основного цикла программы. Заключительная позиция - это позиция солдат в шеренге после выхода из цикла. В этот момент справедливы как инвариант цикла (количество солдат, повернувшихся влево, и количество солдат, повернувшихся вправо, те же , что и в начальной позиции), так и условие выхода из цикла (не существует пары солдат-соседей, стоящих лицом друг к другу). Последнее условие эквивалентно утверждению о том, что каждый из солдат, повернувшихся влево, стоит в шеренге левее каждого из солдат, повернувшихся вправо (убедитесь в этом самостоятельно). Объединив эти два условия, получим сформулированное выше утверждение о свойствах заключительных позиций солдат в шеренге. Наше доказательство вновь иллюстрирует общее правило: если мы хотим формально доказать, что после выхода из цикла справедливо некоторое утверждение (будем называть его *постусловием*), то нам надо поступить следующим образом.

Метод доказательства справедливости постусловия

1. Определить инвариант цикла.

2. Доказать завершимость цикла.

3. Показать, что из одновременного выполнения условия - инварианта цикла и условия выхода из цикла следует справедливость постусловия.

В заключение заметим, что наша программа, моделирующая поведение новобранцев в строю, является одновременно программой сортировки в порядке возрастания массива, состоящего из нулей и единиц. После небольшой переделки ее можно использовать для сортировки массива, содержащего элементы из произвольного множества, на котором определено отношение порядка (хотя это и не лучший алгоритм сортировки).

Задачи для самостоятельного решения

1. Докажите, что для шеренги из n новобранцев повороты завершатся не позднее, чем через (n - 1) секунду.

2. Новобранцы построены в m шеренг по n человек в каждой. По команде "напра-во" одни из них поворачиваются направо, другие - налево, третьи остаются на месте. Спустя секунду каждый из них поворачивается так, чтобы занять ту же позицию, что и сосед по строю, на которого он смотрит (если перед глазами новобранца никого нет, то он своей позиции больше не меняет). В дальнейшем новобранцы действуют по этому же правилу. Напишите программу, моделирующую поведение такого строя. Верно ли для него утверждение о том, что повороты обязательно прекратятся? Если да, то как можно описать множество конечных позиций строя? Если нет, приведите контрпример - начальную позицию строя, для которой повороты будут продолжаться бесконечно. Попытайтесь выявить общие свойства всех таких начальных позиций.

3. Условия те же, что и в предыдущей задаче, только к новобранцам, повернувшимся в начальный момент направо, налево и оставшимся на месте, добавляются повернувшиеся кругом.

С.Рамазанов

Компьютерный бизнес в СССР

История его возникновения достаточно коротка и ведет свой отсчет с 1985 года. Провозглашенная перестройкой демонополизация внешней торговли вывела на мировой рынок сотни кооперативов и совместных предприятий, которым нужно было с наибольшей выгодой вложить имеющуюся валюту в какой-нибудь товар. По целому ряду причин этим товаром стал персональный компьютер, что и вызвало "компьютерный бум" 1985-1989 годов. Перевод валюты в рубли по схеме "валюта-компьютеры-рубли" позволял добиться коэфициента 1:60.

Результат бума - появление на столах советских производственников и бизнесменов персональных компьютеров (ПК) стандарта IBM PC. Если раньше слово "компьютер" ассоциировалось у нас с машинным залом, где по панелям бегают разноцветные огоньки и бесшумно ходят люди в белых халатах, то сейчас любой первоклассник растолкует вам, что такое ПК. Можно уверенно сказать, что персональный компьютер вошел в нашу жизнь всерьез и надолго.

Попробуем повнимательнее разобраться, что за компьютеры появились в СССР и как складываются на сегодняшний день спрос и предложение. По типу практически все попадающие к нам ПК относятся к различным клонам IBM PC, построенным на микропроцессорах фирмы Intel. Создается впечатление, что кроме PC других персоналок просто не существует. Перефразируя слова поэта,

Мы говорим ПК
Подразумеваем PC ...

Кстати, в Европе и США превосходство PC-совместимых компьютеров не столь велико. На рынке машин средней производительности с ними успешно конкурируют компьютеры Macintosh фирмы Apple. На рынке машин высокой производительности (у нас их называют рабочими станциями) доля PC-совместимых компьютеров (старшие модели на базе i80386, i80486) вовсе незначительна - около 7%.

Почему клоны IBM PC захватили наш рынок? Из множества причин выделим две: распространенность и цена. Их много, они универсальны и нам по карману.

Большинство ПК, поступающих в СССР, изготовлено в странах Юго-Восточной Азии (Тайвань, Сингапур, Корея). Реже встречаются компьютеры, изготовленные в Европе и США. Среди советских бизнесменов, занимающихся продажей компьютеров, появились даже термины "желтая" и "белая" сборка, обозначающие страны производства. Сейчас приходится слышать и о "красной" сборке, как вы понимаете - это компьютеры, собранные в СССР.

Разделение ПК на "белые" и "желтые" не так уж безобидно, как это кажется на первый взгляд. "Белый" ПК стоит, как правило, на 2-3 тысячи дороже, чем "желтый" такой же конфигурации. Считается, что компьютеры, собранные в США или Европе, имеют более высокое качество, хотя с этим утверждением можно спорить. Более вероятно, что разница в

цене обусловлена высокой стоимостью рабочей силы в развитых странах и, следовательно, большей себестоимостью компьютеров. Тяга советских покупателей к "белым" компьютерам объясняется скорее психологическими причинами.

Постепенное насыщение рынка, введение драконовских таможенных пошлин и развитие конкуренции лишили персональный компьютер звания "импортного товара N 1", свели на нет "компьютерный бум", и сейчас уже можно говорить о рынке ПК как о сложившемся, с устоявшимися ценами.

Из чего же складывается цена ПК для конечного покупателя?

Первая составляющая - долларовая цена машины, которая для стандартной AT-286 равна $900-1100 (при переводе в рубли по коммерческому курсу - 33-35 тыс. рублей).

Вторая составляющая - таможенная пошлина (для той же AT-286 это - 4-5 тыс.руб).

И, наконец, третья составляющая полностью зависит от аппетита продавца. Надо отметить, что этот аппетит значительно снизился, и норма прибыли в 20-30% вполне устраивает серьёзных продавцов.

К сожалению, в последнее время появилась еще одна составляющая цены - 5% налог с продаж, введенный Президентом в марте 1991 года. Для стандартной AT-286 он равен приблизительно 2000 рублей. Не задев продавцов, эта сумма была полностью переложена на покупателей.

В результате сложения всех составных частей получаем окончательную цену. Цены московского рынка приведены в таблице 1.

Несколько слов о конфигурации компьютеров, предлагаемых к продаже. Наиболее ходким товаром на данный момент являются ПК типа AT-286. Полгода назад стандартным считался следующий комплект.

Процессор 80286, 12 МГц
Сопроцессор 80287, 10 МГц
Оперативная память (ОЗУ) 1 Мбайт
Винчестер 40 Мбайт
Флоппи-дисководы:
5.25" 1.2 МБайт и 3.5" 360 Кбайт
Монитор и адаптер EGA,
диагональ 14",
разрешение 640x350 точек
Принтер широкий, 9 игл
Манипулятор "мышь"

Сейчас положение изменилось. Покупатель стал более разборчив и требует от своего ПК большей мощности, больших возможностей. Первое требование - повышение тактовой частоты до 16 МГц, второе - замена адаптера EGA на адаптер VGA с разрешением 640x480 или даже 800x600 точек, и, наконец, вместо флоппи-дисковода на 360 Кбайт покупатель хочет приобрести более современный 3.5" дисковод на 1.44 Мбайт. Хотя эти усовершенствования требуют дополнительно 2-3 тысяч (см. таблицу), желание иметь более совершенную и престижную машину обычно пересиливает стремление к экономии.

Сейчас можно говорить о рынке ПК как о сложившемся, с устоявшимися ценами

Надо сказать, что приобретение ПК с наилучшими характеристиками не всегда оправдано. Сейчас большая часть персональных компьютеров используется как "умная" пишущая машинка и/или хранилище информации (например, АРМы "Бухгалтерия", "Кадры", "Склад"). В первом случае вполне достаточна XT-8088 с монохромным монитором, во втором AT-286/12, возможно без сопроцессора. Современный монохромный монитор более эргономичен и стоит дешевле, чем мониторы EGA, VGA или Multiscan. Предубеждение советских пользователей к монохромным мониторам, по-моему, не имеет под собой объективных причин, кроме единственной: на монохромном дисплее не так интересно играть в компьютерные игры. Отсутствие спроса на такие машины привело к сокращению предложения, и сейчас найти ПК нестандартной конфигурации практически невозможно.

Что же ожидает компьютерный бизнес в недалеком будущем? По единодушному мнению экспертов советский компьютерный рынок не насыщен еще и на четверть - так что здесь перспективы достаточно радужны. Характерно проникновение на рынок крупных фирм, начиная с "IBM" и кончая "МММ". Дело идет к созданию разветвленной инфраструктуры сбыта по всей стране, подкрепленной развернутой рекламной кампанией. Но и для мелкооптовых торговцев компьютерами еще ничего не потеряно. Для них в ближайшее время наибольший интерес будет представлять сбыт ПК через универсальные и специализированные товарные биржи. А в дальнейшем им придется не только более быстро, чем крупные фирмы, ориентироваться на требования рынка, но и оказывать своим покупателям дополнительный комплекс услуг, начиная с наладки и кончая сервисным обслуживанием.

Таблица 1. Цены на компьютеры в Москве

Конфигурация	Цена, тыс. рублей
XT-8088	35-38
AT-286/12	42-45
AT-286/16	45-49
AT-386/25	108-120
AT-386/33	120-150

Неуловимый Чуча

Д. Кузнецов

Но раз я ее выдумала,
Значит, она есть!

Пеппи Длинный Чулок

Неприятности, как обычно, начались с понедельника. Ира включила машину и увидела ядовито-зеленые буквы: **"ХОЧУ ЧУЧУ"...**

- Ой, что это?

- Вирус! Нажимай контрол-брейк! Нет, лучше выключай!

Вова вставил аварийную дискету и включил машину. Запустили вакцину. Пришел Тема и попросил показать вирус. Стали сканировать дискеты. На нескольких обнаружили бутовый STONE. Кроме того, "Вакцина" сказала, что ibmbio и ibmdos на винчестере "модифицированы". Их пришлось скопировать с аварийной дискеты. Система перестала грузиться... Пришел Саша и объяснил, что системные файлы будут работать, только если записаны на определенном месте. И скопировали файлы на эти места. Пришлось подтвердить, что следует писать именно по абсолютному адресу и именно в область, защищенную от записи. Файловая структура оказалась разрушенной. Ира увидела, что ее тома нет, и пошла по институту жаловаться, что гадкий Чуча съел ее статью. Тема пытался отловить Чучу, просматривая куски памяти визуально. Про себя он решил не вставать, пока не найдет причину. Впрочем, довольно скоро, вынимая аварийную дискету, он обнаружил, что отвалилась позолоченная бумажка, защищавшая ее от записи. Ловить Чучу, пользуясь зараженной дискетой, было глупо, а списать хотя бы основные системные файлы было не у кого (все уже разошлись), и Тема отступился. Наутро невыспавшийся Тема нарычал на редактора стенгазеты, когда тот попытался понять, как Чуча сумел отодрать с дискеты протектор... Парню, который хотел переписать покер с раздевающимися красотками, сказали, что девочки, возможно, заражены, и он решил пока воздержаться от интимных информационных контактов. Студентов предупредили, что релаксации после лекции по философии не будет. Обычно релаксация состояла в том, что они летали на F-15 над Кольским полуостровом и сбивали советские МИГи. (Действительно, другим способом прийти в себя после ушатов диаматерной идеологии трудно.) Специалист, которому показали дискету, сказал, что, кроме всем известного бутового вируса, никакого Чучи на ней нет. Он посмеялся над попыткой писать системные файлы по абсолютному адресу. Оказывается, надо было запустить 'Norton Disk Doctor'. У него в меню есть специальная опция 'Make the disk bootable'. Получилось, что Чуча ловко использовал безграмотность охотников и создал на диске непроходимые дебри.

Тема счел, что к информации на винчестере пора применить "высшую меру социальной защиты", и переформатировал его... Институтский поп-журнал "Резонатор" опубликовал интервью "Таинственный Чуча", а стенгазета поместила аналитический обзор "Чуча у нас дома". Основной мотив - что надо стереть все программы с голыми красотками, потому что через них в компьютеры проникает СПИД. В это время в Институте проходила международная конференция. Глава корейской делегации пришел в комнату, где видели Чучу. Удалось понять, что он прочел заметку в стенгазете и благодарит за поддержку борьбы за нравственность. Еще указал на опечатку: следовало писать не Чуча, а Чучхе. По институту прошел слух, что Чуча - агент Пхеньяна, и все стали лихорадочно запускать все антивирусные программы, какие только удалось достать. Назревал международный скандал... Пришли студенты. Узнав, что винчестер переформатирован, они хотели восстановить антивирусную защиту с помощью специальной программы "Феникс", которую они поставили в прошлый раз, не снабдив соответствующей рекламой. Так вот оно что! Старая вакцина считала файлы, обработанные Фениксом, зараженными!

- "Но ведь Чуча-то была!" - закричала Ира. Студенты смутились. Они признались, что это шутка. Коротенькая программа смотрела, какое сегодня число. Если оказывалось, что 1 апреля, то она стирала свой выполняемый модуль, а потом спрашивала про Чучу. В случае ответа "Чуча" она прекращала свою работу (исчезая таким образом навсегда), а в случае иного ответа повторяла вопрос... Студентов хотели выгнать, но пожалели: с одной стороны, благодаря Чуче (то есть, простите, Чучхе) удалось поймать STONE, а с другой стороны, студенты обещали принести игру, где ни в кого не надо стрелять, а нужно, наоборот, строить город. Тем более, что теперь все программы защищены Фениксом, и никакие Чучхе нам не страшны... Мы уверены в Фениксе. Поэтому нас не пугает, например, что по понедельникам компьютер почему-то стал играть: Никакой на свете зверь, дикий зверь, страшный зверь. Не откроет эту дверь, эту дверь, эту дверь. Да...

Софт · Москва

СПРАВЕДЛИВЫЕ ЦЕНЫ – ОСНОВА НАШЕГО БИЗНЕСА

Компьютерная биржа "Софт–Москва"
и торговая группа "Элк"
предлагают все необходимое для работы

- компьютеры XT/AT–286/386/486;
- принтеры, плоттеры, сканнеры, дигитайзеры, стриммеры;
- кстероксы, телефаксы, модемы;
- телефонные аппараты.

Справедливые цены–основа нашего бизнеса. Обширная информационная база позволит предложить Вам цены, соответствующие реальному качеству товара.

Для оптовых покупателей предоставляется скидка.

Для розничных покупателей полный комплекс работ по тестированию и пуско–наладке.

Для всех – бесплатные консультации.

ЖДЕМ ВАШИХ ЗВОНКОВ: (095) 236–96–13
ПИСЕМ: 117419, Москва, а/я 765.